Dynamic Editable Models of Fire From Video

Chinery, Andrew

Award date:
2015

Awarding institution:
University of Bath

Dynamic Editable Models of Fire From Video

submitted by

Andrew Chinery

for the degree of Doctor of Philosophy

of the

University of Bath

2015

Summary

This thesis presents the following statement: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* Current techniques allow flames to be modelled via physical simulations; but these methods require significant effort to produce content, and due to unintuitive parameters it is difficult to produce a specific visual result. Alternatively, models can be acquired from video; but only as a density field representation, and this is not editable.

In this document, a novel method for modelling flames is introduced, the 'flame core model'. This is approach consists of two parts: a structural element – the core – and the density that surrounds it. The parameters of this model are intuitively editable and can be found from three-dimensional video data. Using these, new flames can be generated from existing sequences. This significantly simplifies the problem of content generation – from the laborious task of a skilled user, to the simple act of filming video of a real flame.

Some flame appearance qualities can also be learned from simple examples. With this technique, it is possible to use a single frame of a single uncalibrated camera to provide parameters for the appearance of a three-dimensional moving flame, something that is completely novel for flame reconstruction literature. The video data filmed for this project and the three-dimensional reconstructions are also state of the art for the field.

Acknowledgements

First I owe a great amount of gratitude to my supervisor Peter Hall. His intuition for statistics and problem solving in general made this work possible. I have learned a great deal in this process but do not come close to his genius. He helped me until the end even through busy times, and my work is so much better for it.

To my mum and dad, who have given me everything I have. Thank you for always being there when I need you day or night, and I'm sorry I always leave it far too long without calling. I couldn't wish for more supportive parents.

Next, a world of thanks to Tom Lovett. For all the help he gave me understanding the work of others, the work I had done, and the work I should be doing. For the last minute proofreading and just generally making time to help anyone in need. It is genuinely a great loss that academia could not hold on to you.

Thanks to Alexander Johnson who gave me the gift of opera singing, making me a more confident and able person in the process, and introduced me to the most beautiful music in the world. And to Guy McCusker, my undergraduate supervisor, who introduced me to research and continues to be an inspiration with his breadth of knowledge.

Thank you to everyone who has contributed ideas or feedback about this work: to Phil Willis, Darren Cosker, Matt Brown, Chris Li, David Pickup, and Tom Saunders for the early meetings, and to Dmitry Kit and Luca Benedetti for their help with capture and for taking the work further. Also thanks to everyone who supported me while I was lecturing: Joanna Bryson, Darren Cosker, and Brian Wyvill on campus, and to Andy Bashford and Liz Shortridge for giving me a place to stay and looking after me. Further thanks to Tom Fletcher and Ieuan Evans for housing me in the last few weeks.

Thanks to all my colleagues for making day to day life fun: Andy Ridge, David Wilson, Ryan Kelly, Michael Wright, Dan Gooch, Jim Grimmett, Lizzi Gabe-Thomas, Alastair Barber, and anyone who ever came for coffee with us.

Finally thank you so much to the most important person in my life, Kate Aldridge, for supporting me in every way imaginable.

Contents

List of Figures

Chapter 1
Introduction

This document presents a novel approach for modelling flames. It allows for visually intuitive modifications, can be driven by data captured from real flames, and can synthesise new flames which are unique but visually similar to the original. This is the hypothesis of the work: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* Figure 1-1 shows an overview of the process from real video to synthesised frames of a novel animation. As later chapters will show, the results look realistic – particularly in motion, are visually similar to the original flames, and are intuitively editable.

1.1 Background

The word fire refers to the result of the chemical process of combustion, which is the rapid oxidisation of a flammable material, releasing heat and light. The flame, the focus of this work, is the visible portion of the fire. The typical yellow appearance of a flame is due to many glowing soot particles that are produced in the reaction. Computer models of fire are studied for many applications, ranging from accurate health and safety oriented physical simulations, to graphical techniques that push the boundaries of the available hardware. In the entertainment industry fire is abundant, there is rarely an action film or a game which does not feature it in some capacity.

Computer simulations of flames can be used to create visually appealing animations, as shown in Chapter 2. They can achieve this by physically simulating the interaction of the burning material (often a gas), including soot particles and other products for full effect.

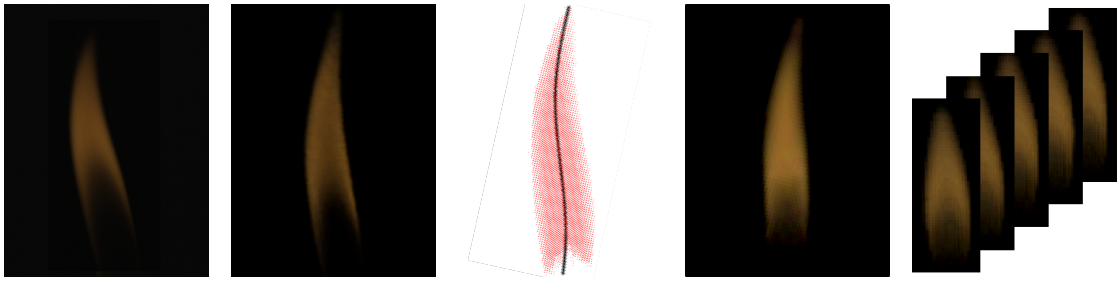


Figure 1-1: Using real video to create a new flame animation, via an editable model

However, the actual combustion process is

- extremely complicated – one chemical kinetics scheme, GRI-Mech, composes 325 elementary chemical reactions to describe the burning of biogas [SGF+12]; and
- chaotic – small changes in the parameters will have wildly different results in the appearance of the flame [LF02].

Both of these mean that simulations are hard to control to create a desired result. The complexity means that the computation time is long, and this alone makes tuning parameters difficult. Coupled with the fact that changing one parameter can undo the effect of tuning another, physical simulations are not suitable for most production environments.

Another approach is simply to film the fire. Pyrotechnics have been used for thousands of years, most likely originating in China or India where they were used for military advantages [CM10]. Now they are common in stage productions and firework displays; so their safe use in proximity to people has been extensively researched [Urb06]. Many filmmakers will use real pyrotechnic effects for 'important' or visually salient fire and explosions on camera. They then film more elaborate and dangerous examples, and composite these into the scene digitally. Rather that film it, it is common for a filmmaker to buy footage of this sort from a company who specialises in filming the effects safely. It is possible for this technique to create very convincing results. But the complex nature of fire means that an expert user is required to actually composite the scene. Humans are naturally good at noticing when small details are not right in natural phenomena, so this makes it particularly difficult to do convincingly.

Techniques exist which go part way to bridging the gap between these approaches. Video itself can be used as a form of control for a digital model of a flame. However, once it has

been created the user has no control over the result – other than to directly edit values in the way one would edit a photo. This is particularly unintuitive in three dimensions.

1.2 Motivation

The current methods of generating flames do not offer the best of both worlds. It is possible to create a user-controllable flame model, but the controls are extremely complicated and only suitable for an expert user. It is possible to use video directly in a scene; but it must be filmed from the correct angle, and is difficult to composite convincingly – both of these also require a skilled user, and the video cannot be reused. Video can be used as a form of control to drive a three-dimensional flame model, however the current techniques provide only a simple representation; they cannot be controlled beyond what is in the input video, and they cannot be used to generate similar looking flames or provide other analysis. Furthermore, these methods are not simple for an artist either – the video is required to be filmed in a way that is impractical.

A system that could take video input, give further control in an intuitive way, and generate flames that are realistic, would be a significant contribution.

1.3 Hypothesis

The hypothesis this document asserts is the following statement: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* This is elaborated and given more precisely in the following sections.

1.3.1 Objective

The main objective of this work is to create a method for generating intuitively editable models of flames from video footage. 'Intuitive editability' in this instance means that the parameters of the model should have direct and obvious links with the visual representation of the flame. Hence it will be easily usable, without having to understand the underlying real world properties of the flame. Two examples of models that are not intuitive are: a model in which the parameters are non-orthogonal, that is, changing one parameter can undo changes that were made to another parameter. Or, a model in which the artist is

required to alter minute elements of the model directly, such as changing individual pixels values in an image.

The rendered results should foremost look visually convincing as flames, and secondly strongly resemble the input video. They will be three-dimensional, and animated. These are both essential qualities of realistic looking flames, and significantly broaden their potential for application. Further, this should enable the creation of a method for generating new flames that are unique but visually in the same style of a given flame. This includes but is not limited to: colour, shape, and the appearance of the motion. There are no specific technical restrictions on the input video itself. However, the less complicated the video acquisition process the better; and the capability to generate flames based on uncalibrated single view videos is a goal that would be a great contribution.

1.3.2 Challenges

There are many challenges associated with the objectives.

1. Creating a flame model which is specified only by intuitive parameters.
2. Rendering realistic looking flames from these models.
3. Obtaining good quality video footage of flames to use as input.
4. Finding parameters that create a flame visually similar to the video, particularly when working from two-dimensional input.
5. Generating arbitrarily long, new sequences of flames that resemble but are distinct from the original.
6. Enabling control over this process, allowing the distinctions to be specified in a visually intuitive way.

1.3.3 Contributions

There are three distinct notable contributions that have resulted from this work. The first contribution is a model of flames that is intuitively editable and can be fitted to input flame videos. This is presented in Chapter 4. Previous work in this field is on complicated digital models from scratch, or simple reconstructions from video. Both are

hard to control. This model gives realistic looking flames that can be controlled by an input video, and then further modified using intuitive parameters. The model is designed to fit to three-dimensional flame reconstructions, however a method is also presented for learning parameters from a single uncalibrated camera, something that has never been done before in flame reconstruction. This is given in Chapter 5. With the visual intuitiveness of the model, this enables simple modifications or parameters from a single two-dimensional image to be applied to an entire library of three-dimensional flame shapes, creating brand new flames extremely simply. This satisfies challenges 1, 2, and 4.

The second contribution is a generative method which can create novel flame animations from existing ones, and this is detailed in Chapter 6. This novel procedure produces an arbitrary number of frames which match the original in style – motion, shape, and colour – but are not identical. This enables existing animations to be extended indefinitely, allows for new animations to be formed, or for flames to be cloned many times in a scene and all look similar but not the same. The generative process also has intuitive controls, allowing specific new types of flames to be formed; or entire videos can be created by merging a single fitted flame with the other properties of a full animation. This satisfies challenges 5 and 6.

Finally the input video used to demonstrate the results was filmed specifically for this project and is a significant improvement over any other examples available in the field. It consists of two long sequences of frames from multiple synchronised cameras. High resolution three-dimensional voxel reconstructions for this data, and the other available source video, are provided as well. This work is detailed in Chapter 3, and satisfies challenge 3.

1.4 Document Structure

This document is structured in the same order that is necessarily to create the end result, since later work builds off earlier concepts. This chapter gave an introduction to the concepts, then

- Chapter 2 reviews the background work that has been done in this field;
- Chapter 3 documents the results, procedure, and technical details of the newly captured flame data;

Figure 1-2: Images of real fires, demonstrating naming conventions
Left: fire made of many flames
Middle: flame with two distinct licks
Right: flame with a single lick

- Chapter 4 details the novel model used to represent the flames;
- Chapter 5 builds off 3 and 4, explaining how models can be found from real video data;
- Chapter 6 presents a method for generating new animations from a sequence of flames found using Chapter 5; finally
- Chapter 7 gives the conclusions of the work, and areas for potential future work.

1.5 Nomenclature

For the rest of this document the following naming conventions will be followed:

- The term *fire* encompasses every element of the natural phenomenon. It is a fuel undergoing rapid oxidisation, releasing light and heat. A fire can be big or small.
- *Flames* refer specifically to the *visible* portion of a fire, it does not refer to the fuel but the volume which is perceived. Furthermore, a flame is usually a single stream, from a single source. A forest fire, for instance, would create many flames, whereas a burning torch would create one flame.
- A single flame can still create many *licks*, which seem to split and merge as the flame flickers. These licks still emanate from a single source, but they have a separate structure.

These terms are depicted in Figure 1-2.

1.6 Mathematical Notation and Conventions

This document uses consistent typesetting conventions for mathematical notation.

Scalars are given as lower case italics. e.g.

$$a,\ x,\ \delta \tag{1.6.1}$$

Functions have lower case italics with parameters or are clear from context. e.g.

$$c(u),\ x\colon \mathbb{N} \to \mathbb{R},\ \xi\colon h \mapsto [0,1] \tag{1.6.2}$$

Vectors are lower case bold characters. e.g.

$$\mathbf{v} = (v_1, \ldots, v_N),\ \mathbf{m} \in \mathbb{R}^3,\ \boldsymbol{\psi} = (\psi_1, \psi_2, \psi_3) \tag{1.6.3}$$

Matrices are italic upper case characters. e.g.

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ M = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \tag{1.6.4}$$

Traditional sets of numbers are given in black-board font, as usual. e.g.

$$\mathbb{N} = \{1, 2, 3, \ldots\},\ \mathbb{Q},\ \mathbb{R} \tag{1.6.5}$$

All other sets (actually multisets unless specified otherwise, see below) are given as bold upper case characters. e.g.

$$\mathbf{F},\ \mathbf{L},\ \mathbf{D} = \{\mathbf{p} | p_z > 5\} \tag{1.6.6}$$

1.6.1 Usage of Sets

This document refers to collections of elements which are unordered, but differentiate based on the multiplicity of the elements. In other words

$$\{1, 2, 1\} = \{1, 1, 2\} \tag{1.6.7}$$

but

$$\{1, 2\} \neq \{1, 1, 2\}. \tag{1.6.8}$$

For this purpose an indexed family is too strict (it is ordered) but a set is too general (it obeys the axiom of extensibility). Instead, these are represented by multisets [Knu69, p. 694]. This is formally defined as a 2-tuple (S, m) where S is a traditional set and $m\colon S \to \mathbb{N}$ is a multiplicity function which gives the number of times each element appears in the set.

In this document, unless specified otherwise, set notation should be assumed to refer to a multiset. For example, the multiset $\{1, 1, 2\}$ could be formally written $(\{1, 2\}, m)$ where $m\colon \{1, 2\} \to \mathbb{N}$, $m(1) = 2$, $m(2) = 1$.

1.7 Collaboration

The work in this document was completed without any significant collaboration. Many ideas, inspirations, and explanations were provided by the author's supervisor Professor Peter Hall, and other colleagues at the University of Bath (detailed in the acknowledgements).

1.8 Implementation

All experiments were implemented in Matlab 2012b (8.0) and run on a MacBook Pro Retina (2.3GHz Intel i7, 8GB memory) and/or a Windows 7 desktop (2.93GHz Intel i7, 16GB memory).

Chapter 2
Background Material

In this chapter the background material that is relevant to the hypothesis is detailed and discussed. That is, *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* Since the objective is a graphical model from video, the most relevant prior works for this project are from computer graphics and computer vision literature. These are discussed, but first a simple explanation of the chemistry of fire is provided to give context to all future material.

The main conclusion from this chapter is this: *model acquisition is difficult and expensive.* There exist flame models which give realistic results and control over appearance, however, the control is complicated, and this means that it requires a skilled user and time to create flames. There also exist methods for producing models from video examples, however these are not controllable.

There is a gap in the literature for a method which can produce controllable flame animations from video. This would simplify the problem of acquiring models, from a time-consuming activity that requires a skilled artist, to just filming the desired effect.

2.1 Fire Chemistry

We understand inherently what fire does and how it behaves; however when one thinks about what the composition of fire really is, the answer is not necessarily clear. Before computer representations are discussed, the underlying physical properties and chemistry of fire will be established. This will be useful later for reference and inspiration.

The most basic description of fire is a reaction between fuel, oxygen, and heat. Slightly

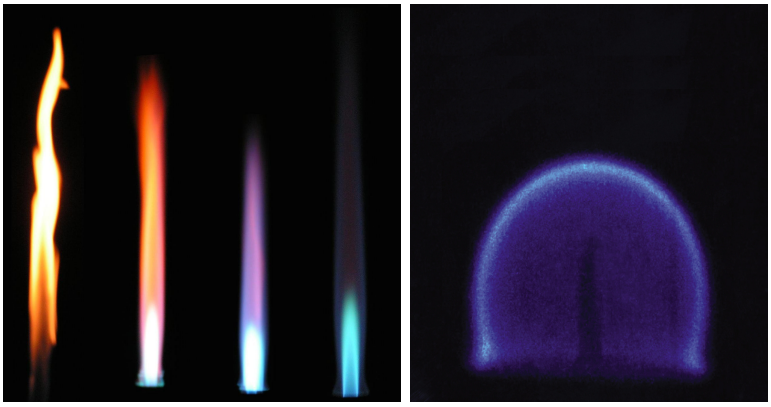


Figure 2-1: Left: states of a Bunsen Burner with differing levels of oxygen
Right: a spherical flame in zero-gravity

more complicated is to specify that the fire itself is the rapid oxidisation of the fuel, caused by heat, which produces light and more heat. More technical still, a fire requires two or more components which combine when heated. While it is theoretically possible to have other oxidising materials, in everyday life one of the two elements is always oxygen. If the other component – the fuel – is an organic solid (e.g. wood, paper, or some plastics), then it must first be converted into a gas. This is done by applying heat, giving the fuel more energy, resulting in a mix of gas, liquid, and solids with higher carbon content called char. This process is called pyrolysis. Then, the gas fuel, still excited from the heat energy, reacts with the oxygen in the air. For example, in a simple flame such as that of a candle, the wax breaks down into carbon and hydrogen, and these react with the oxygen giving carbon dioxide and water vapour respectively. These reactions give off blue light, and this is called chemiluminescence. Normally a candle flame is yellow, and this is due to an excess of carbon atoms in the fuel. There is not enough oxygen for them all to react, so instead they appear as soot in the flame. Since they are hot, they glow yellow, and this is called incandescence. This is why a Bunsen burner shows different states depending on the amount of oxygen allowed to mix with the gas (shown in Figure 2-1, left).

There are two things associated with a flame: heat and light. In a clean flame, one with little soot, the reaction that is producing the light is happening on the surface of the volume perceived to be the flame. This is where the heated fuel is reacting with the oxygen-rich air. The heat is being produced on the surface as well, however it is spreading, meaning the centre containing just the gaseous fuel is hot as well. In a sooty flame these particles are spread throughout the flame, so light emits from the centre as well. The more yellow the flame, the more of the light is being contributed by soot particles. In any flame the

majority of the light still emanates from the surface of the volume, as this is where it is hottest, and hence the particles are brightest.

The spreading heat is what continues the reaction. There are three main ways in which heat is transferred: conduction, convection, and radiation. Convection in particular plays an important role in fire, as it is what gives a flame its distinctive tear-drop shape. When air is heated, the particles gain more energy. More energy means that the particles get excited: they spread out, meaning that the air has lower density. Lower density means in the presence of gravity it will rise (as gravity will cause the more dense air to sink). The air around the base of the flame is being heated, this causes it to rise and to be replaced with new (colder) air, and so the process repeats. The hot gas inside the flame is rising as well. This is why an undisturbed flame from a single source is tall and thin, the air is rising around the outside of the flame until no unoxidised gas remains at the top, or the reaction has cooled to a point where it does not self-propagate. This was experimentally confirmed when NASA demonstrated that a flame in zero gravity takes a spherical shape [RWPW98] (shown in Figure 2-1, right).

2.2 Computer Graphics

Moving to digital representations, first the tools for creating and animating flames directly are described. These fall into the field of computer graphics. One objective, from Chapter 1, is that the model must be intuitively editable, so it is relevant to look at current techniques that are used by artists.

2.2.1 Two-Dimensional Video

Many applications (especially those with lower budgets) will simply use flat two-dimensional videos of flames that are composited into the scene. Entire libraries of fire video can be purchased for this purpose. However, without considerable effort it is unlikely the result will hold up to visual rigour, due to the inherent three-dimensional nature of flames. This is because as soon as the camera rotates, the two-dimensional image will be become apparent; rotation of the camera is regularly employed in film making and often user-controlled in video games. Some solutions rotate the flame so it is always facing the camera, but this can still be recognised easily. Furthermore the control over these images is very limited, typically the only modifications that the artist can make are to the size, colour, orientation,

Figure 2-2: Fire Studio 5 uses two-dimensional flames in three-dimensional environments

or other simple modifications which can be applied to an entire video but do not take the flame itself into consideration. Plus, creating an effect where the flame reacts to the scene, or vice versa, is extremely difficult in this situation, since they are essentially generated independently. Figure 2-2 shows a screenshot of Digital Combustion's software Fire Studio 5, which uses two-dimensional flame videos that are composited into a scene with added simulated smoke. Even from a static screenshot the flames are clearly two-dimensional and not convincing.

When visual realism is required at any cost, a director may film flames and explosions separately to the scene but in exact (or scaled) camera conditions. The ability to film with this precision requires high expertise, and then compositing this with the scene requires another skilled user. This approach is inaccessible to most users due to cost and skill requirements. Furthermore, it cannot be used in situation where the viewpoint of the camera may change arbitrarily, such as a video game.

2.2.2 Three-dimensional

In many situations, full three-dimensional flame models are more useful. Precisely how these flames are represented varies between applications. Ultimately, the only works of concern are those that produce flames that can be displayed on a screen, as opposed to work designed for the study of properties of flames for health and safety, for example. Graphical approaches usually consist of a model, and a rendering technique that produces

an image from that model. Often, flames are represented as density fields; that is, some function $f(x, y, z) \mapsto \mathbb{R}$ gives a density brightness for any point (x, y, z). Then these can be rendered using a technique like volume ray tracing [DCH88]. Or they can be displayed with a real-time technique like texture mapping and rasterisation [HS89b]. The densities can be quantised into cubic areas, and these are called voxel models; they are essentially three-dimensional images with transparency. Some techniques produce voxel models as their output, which can then be viewed from arbitrary camera angles or rendered into a specific scene using the previously mentioned techniques. Others produce particle systems, discussed next.

2.2.3 Particles

A very common technique is particle systems, first presented in [Ree83]. These can be used to describe a great variety of phenomena in addition to fire, such as water, explosions, dust, hair, fur, or abstract visual effects like magic spells. Particles can be thought of as tiny balls which have several parameters including colour, transparency, and lifespan. They are produced in great numbers at the surface of a particle emitter, which is typically a simple mesh object such as a plane or a cube. At production time, the parameters of the particles are set, including some physics model for the particles to move. This could be governed by a global physics model (such as wind), the particles could each be given parameters that drive their motion (e.g. velocity and acceleration), or it could be a combination of the two. Typically, all of the parameters will have some 'fuzziness' to them. In other words, there will be random variations within a controlled range, which increases the realism of the entire system, in some cases literally making edges appear fuzzy or blurry. A particle system, once created, has two steps: update and rendering. In the update step, the parameters of each particle are updated according to the applicable rules. In the rendering step, an image is produced of the scene. There are several approaches to rendering a particle system, the most basic is to treat each particle as a simple light emitter [Ree83]. Better results can be obtained in real time by treating each particle as a textured billboard [MBGN98] (a quad that is facing the camera). Offline approaches include using metaballs [WT90] or replacing each particle with a mesh and rendering using another standard technique.

Particle systems can be very cheap in terms of computation and storage, because simple rules govern their generation and no extra information needs be stored except the emitter and parameters. Fire that has been created using particle simulations can look quite

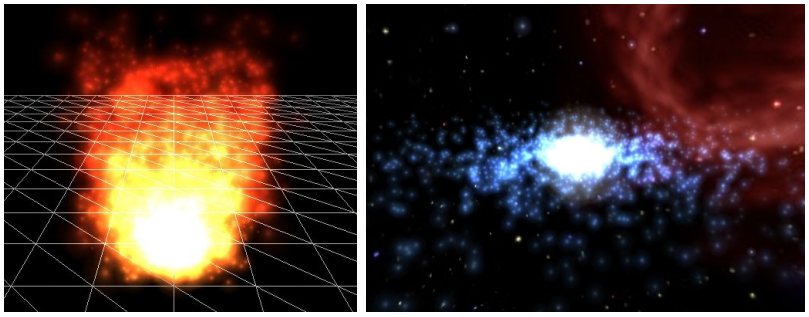


Figure 2-3: Particle simulation renderings of fire and a galaxy

realistic if created by a skilled user, however they can often end up looking like orange fountains. Usually images of real fire are used as textures for the particles, but it is only as good as a colour palette, the actual appearance of the real flame is not necessarily conveyed. Furthermore, it is not very intuitive to an artist how to tune such specific parameters in order to achieve a more realistic result: while it is reasonable for a user to set a colour range, trying to work out what physical movement parameters will produce a flame of a certain shape is more difficult. Figure 2-3 shows examples of scenes rendered using particle simulations.

2.2.4 Physical Simulations

While particle systems are often referred to as simulations, they only really simulate the behaviour of a construct designed entirely for appearance, and are not directly based on any real world physical object. However, such simulations do exist for fire. As mentioned previously, fire simulation has received much attention for the purposes of health and safety; the ability to model how fire spreads in an urban environment has a huge contribution to the process of planning buildings. The model developed by the Center for the Simulation of Accidental Fires and Explosions [HMS+00] is intended to provide this accuracy of simulation, especially in the case of handling and storing of flammable materials. This model formed the basis for [PP06], which applies their combustion models to a graphical environment, providing flames which look very realistic, and can even model the difference in burning different materials. Some example results are shown in Figure 2-4.

Simulations also give a good foundation for objects to interact with the flames. The work of Nguyen et al. [NFJ02], which is based on Navier-Stokes equations with added details to capture the complexity of the flame, can model this interactivity. Figure 2-5 shows an

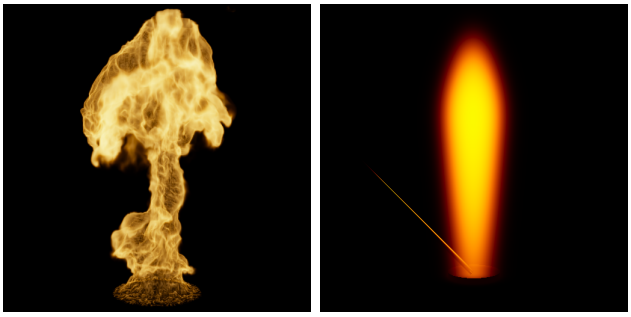


Figure 2-4: Results from [PP06], showing burning methane (left) and sodium (right)

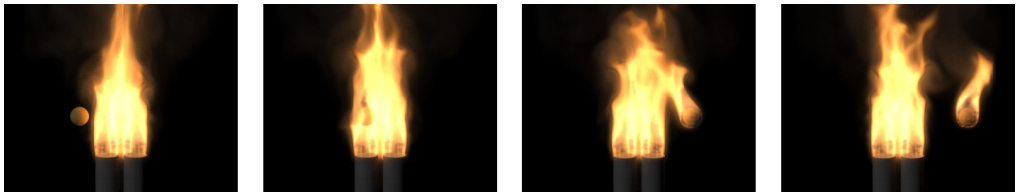


Figure 2-5: A flammable ball moving through a flame from [NFJ02]

example of a flammable ball passing through a flame and catching alight.

Unfortunately, physical simulations are not intuitive for an artist. The parameters required are specific and complex, they are difficult to manipulate even with intimate knowledge of the chemical and physical properties of fire. Furthermore, since they are directly tied to physical conditions, they are not orthogonal. In other words, it is possible for a change in one parameter (e.g. pressure) to undo the careful tuning of another (e.g. wind velocity). They can also be chaotic; changing one parameter slightly can have an extremely different effect on the result. This is necessary for replicating real life conditions, such as simulating a fire in a building. But for an artist working visually, this makes it extremely difficult to iterate towards the desired result, which is aggravated by the high computational cost of rendering: as the resolution increases, complexity increases by at least $\mathcal{O}(n^3)$ [LF02].

2.2.5 Structural Modelling

The shortcomings of simulations are recognised by Lamorlette and Foster, and they suggest an alternative model called structural modelling [LF02]. In this, a flame is represented by a base curve, which goes through an iterative eight-step process which serves to generate additional frames in the animation – accounting for flames splitting and joining, and moving around – as well as incorporating the realistic appearance of the flame as well.

The actual appearance of the flame is created by particles (Section 2.2.3) created inside a volume that is revolved around the curve. The full list of steps are as follows.

1. A particle is generated on the surface of the burning object, and released into a wind field and advanced one time step. Another particle is generated, and the line connecting these forms the base curve.

2. At each frame of the animation, the points are updated for movement resulting from the wind field, diffusion, buoyancy, and the motion of the source. A B-spline is fitted to the points [DB78].

3. Separation and flickering of the splines is randomised based on measurements from video.

4. Particles are generated in a volume defined by a cylindrical rotation of a profile of a flame (either hand drawn or from a photograph).

5. The particles are displaced by Flow Noise to give structural fluctuation [PN$^+$01].

6. The volume is transformed to match the structural curve from earlier parts, and Kolmogorov noise is applied to give small scale turbulence [SF93, SF95].

7. Particles are rendered using a colour palette from an image of a flame.

8. Optional procedural controls allow flames to spread and smoke to be generated.

Step four of the process generates a particle system that has been fitted to a profiled version of the flame spine. However, unlike Section 2.2.3, which describes a system that is completely unbounded and would require a lot of artistic ability and familiarity with the technique, this method encapsulates essential details which give the flame a realistic appearance with no effort from the user. The user is left with defining a wind field, and some procedural controls that govern properties such as intensity, lifespan, and evolution in shape and colour. Contrasting this to the approaches of Section 2.2.1, it is clear that this gives far truer control over the flame itself, not just its pre-rendered appearance. This approach produces attractive results, as shown in Figure 2-6.

However, the method is complicated, and many details are proprietary, since the work comes from the film studio DreamWorks. Several functions are not clearly defined, and the measured statistical data is not available. Furthermore, the approach still requires some unintuitive, non-visual parameters such as generating a wind field or tuning the noise function parameters. This is motivation for considering the use of video to improve flame

Figure 2-6: Structural flames from DreamWorks' Shrek, from [LF02]

models. In this case, the video could set all of the non-intuitive parameters automatically. This could directly lead to better graphical representations, or provide data for analysis to further constrain other methods such as structural flames, to allow the artist to ignore more of the technical details that make up a simulation.

2.3 Computer Vision – Two-dimensional

Relevant computer vision techniques for natural phenomena largely fall into one of two categories: extending video and reconstruction. The former tend to be more general, and hence more applicable to different types of material. Provided the input video has some sort of repetitive quality, these techniques try to learn a model that can repeat and possibly isolate this quality, sometimes adding in random variations to prevent the video from looking like it is just looping. Too extreme motion, or motion which is too irregular, can cause the model to struggle. This can cause artefacts or strange results. The techniques tend to work directly on pixel values, so they are typically non-assuming about their input. For instance, the type of object being filmed and the calibration of the camera may remain unknown. Usually the subject will be three-dimensional, but this is not directly modelled.

Reconstructions, on the other hand, create full three-dimensional representations of the object being filmed. This is a field which is unsolved even for simple rigid objects, so highly dynamic natural phenomena present a challenge. This often means that more assumptions will be made than for simple video extensions. Usually techniques are designed for capturing a specific phenomenon. For instance, you could expect a wildly different result if you are reconstructing water from if you are reconstructing fire. The former is mostly transparent, with spectral reflections on the surface; and as discussed, the latter emits light, and can be represented as a semi-transparent density field. Tuning the model

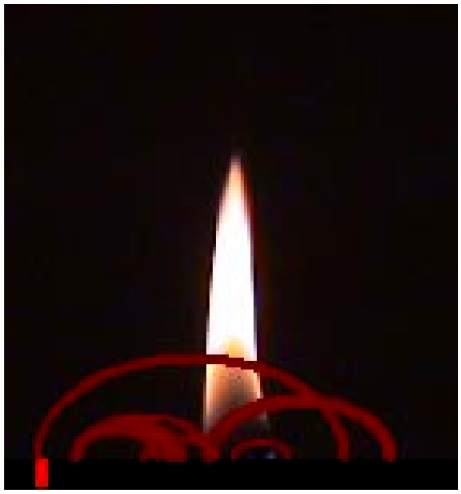


Figure 2-7: Candle flame Video Texture

to best fit these qualities can help constrain an otherwise wildly difficult problem.

Since being three-dimensional is essential to the objectives of this project, reconstruction methods are more applicable. Video extension methods are historically significant and provide useful techniques, so a few examples are briefly discussed.

2.3.1 Video Textures

An early and very popular technique for extending video is Video Textures [SSSE00]. These are designed to produce a video clip that could be used to replace a static photo to give some dynamic qualities. The method repeats frames, and short sections, but never the entire sequence. The basic approach is to find places in the video where it is possible to jump to another section without any visual discontinuity. This leads to little loops of video which are repeated, and can be randomised in order to reduce the obvious effect of playing the same section of video again and again. Each red arc in Figure 2-7 is a link between two frames of a flickering candle. As the video plays, it will randomly choose whether to make one of these jumps or not (unless it is the last jump). This means that the same patches of video do not just get played over.

The method has to make sure that motion is preserved. For instance, in a video of a pendulum swaying left and right, a frame where it is swinging to the right is likely to match one where it is swinging left. This would result in jarring motion, so subsequent frames can be considered to prevent these transitions. Also, videos of very complex structures

may not have frames that are visually close enough that a jump can be made without looking jarring. For this, it is possible to morph between the frames, which causes a small amount of blur but not enough for a typical user to notice.

Video textures are very useful for creating looping videos where a 'non-looping' quality is desired, but does not need to hold up to intense observation. No underlying model that represents the dynamic qualities of the object being filmed is learned, it just uses short clips from the original video. This means that after watching for a while it will become obvious that the video is looping, even if not strictly linearly.

2.3.2 Dynamic Textures

Doretto et al. introduced the term Dynamic Textures in their pioneering work of the same name [DCWS03] which learns the statistics of an input sequence in order to generate entirely new frames which show the same dynamic qualities. Taking the concept of a static texture and extending it into time, they note that sequences of images are clearly not independently drawn from a simple distribution, because there are underlying dynamics which drive coherence between the frames.

They model a general dynamic texture as the output of an auto-regressive moving average process (ARMA), which is essentially a random process, but still drives this temporal coherence in a dynamic texture. Furthermore they give a closed-form solution for learning the parameters of this model from a sequence of input images, for dynamic textures they classify as "second order stationary". This means sequences where the second order statistics are time-invariant. One way to imagine this would be a video which could be repeated to form an animation. For example waves would be second order stationary, but an explosion would not. With a learned model for a particular dynamic texture, they are able to synthesise new frames to extend the video. The authors demonstrate the method captures the dynamics of scenes such as waves, fountains, and flames.

In practice, the synthesised frames of the video do show similar dynamics to the input video. However they contain many artefacts that would make the video unusable in most applications, and these tend to build as more frames are synthesised.

Figure 2-8: Synthesised frames (bottom) show artefacts using Doretto et al.'s method (Images taken directly from the paper [DCWS03], emphasis added)

2.3.3 Learning Flames

In a similar area to video extension, the work by Stitch and Magnor [SM05] can generate unique two-dimensional flame animations from existing examples. They introduce a novel model based on a combination of parameters derived from geometric moments [MR98] along the flame, and a set of 'eigenflames'. Both of these are found from the input video. These parameters are used to drive an auto-regressive process, which once learned can generate new sequences of flames.

Like the other methods in this section, this approach is limited to just two-dimensional video. The generated sequences look convincing, and some mechanisms are proposed to allow control over the overall shape by manually offsetting some of the original geometric moment parameters – path and width. These parameters control only the width of the flame, not the appearance of the density, which is driven just by the eigenflame model, and thus harder to manipulate. This method bears some resemblance to the concepts used later in this work, although they were conceived entirely independently. This, as well as the structural modelling method (Section 2.2.5) show that the concept of a structural element along the length of the flame is a natural and established idea.

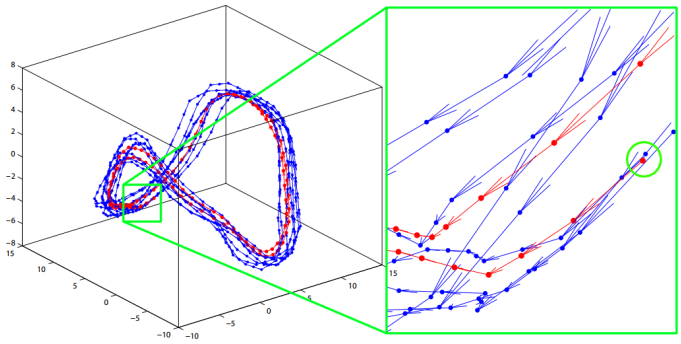


Figure 2-9: A time series embedded in a three-dimensional phase space from [BS09]

2.3.4 Chaotic Modelling

Another technique that can be used for video extension, Time Series Prediction by Chaotic Modeling of Nonlinear Dynamical Systems [BS09] uses concepts from chaos theory in order to capture the underlying dynamics of both human actions and dynamic textures. The idea is that time series can be embedded in a higher-dimension space in a way such that this higher-dimensional representation forms a strange attractor. Given this representation and an initial condition, it is able to predict future states of the system using kernel regression, which avoids making any assumptions about the exact form the function representing the dynamics takes.

In simple terms, imagine that the consecutive points in a time series (e.g. values of a single pixel over a video) form a loop when they are plotted in a higher dimensional space under some embedding transformation. In Figure 2-9 the loop is in three dimensions. Then for an initial value which is close to or on this loop, the next state is predicted by simply moving in the same direction.

A time series is found from sequential pixel values in the video

$$(x_0, x_1, \dots, x_t, \dots). \tag{2.3.1}$$

This is converted to a phase space representation of the form

$$(\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_t, \dots) \tag{2.3.2}$$

where

$$\mathbf{z}_t = (x_t, x_{t+\tau}, \dots, x_{t+(d-1)\tau}), \tag{2.3.3}$$

for some value of the lag τ and the embedding dimension d. This d-dimensional representation should embody the dynamics of the pixel values. Prediction is done in this phase space by taking a weighted average of the motion of nearby points, which gives the next state in phase space. This can then be converted back to the time series representation to give the next image in the video, preserving the texture's dynamics.

The prediction step is quite simple, and works well for simulated data from chaotic attractors. The key to getting good video extension is finding the values for τ and d that give a suitable higher dimensional embedding. The authors use mutual information [FS86] to find the lag τ and false nearest neighbours [Cao97] to find the embedding dimension d. In trying to replicate the authors' work, these techniques were not found to give consistent results over a range of image sequences. However, the authors show good results extending various types of video including flames, and predicting human motion.

2.4 Computer Vision – Three-dimensional

The previous sections on video extension show that it is possible to create a model which encapsulates the motion in a scene well enough to generate new frames for a video. However, this is specific to the video, it could not be applied to another scene that did not originally contain the phenomenon. Most importantly, these methods offer little in the way of control. While Doretto's dyanmic textures technique does offer some control over speed and intensity [DS03b], these are not a large improvement over the controls that are available for an entire video sequence (such as the two-dimensional techniques in Section 2.2.1).

Using a reconstruction method would give a model of the flame independent of its video. These techniques are used in medical imaging for accurate or descriptive representations of data, or in graphical applications for three-dimensional model acquisition. Photometric stereo techniques for reconstructing objects from images were first introduced in 1980 by Woodham in [Woo80]. They make use of Lambertian reflectance over the surface of an object. Put simply, the shading of a surface is based on the angle between the surface normal and the light source. Therefore if the direction of light is known or estimated, and the image shows the shading, then the surface shape can be calculated. However many objects do not obey strictly Lambertian reflectance. Fire, which emits light, clearly does not.

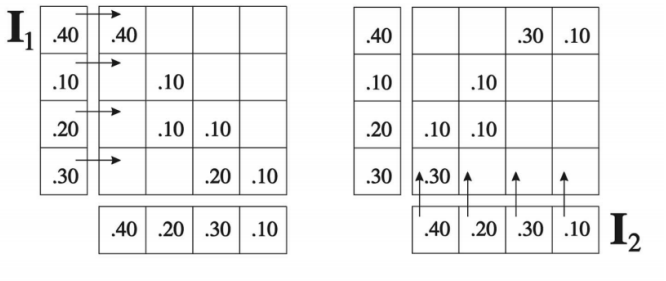


Figure 2-10: Epipolar lines of two views are combined to create a sheet-like solution, image from [HK07]

Tomographic reconstruction, which is used for many medical imaging applications, uses projections from many angles around an object, such as those from an x-ray machine. This form of projection fits the Radon transform (first introduced in 1917 [Rad17]) and by taking the inverse Radon transform it is possible to reconstruct the original density volume. This technique forms the basis for an image-based tomographic method which is discussed later.

A review of fire reconstruction literature is given here. For a comprehensive review of all transparent and reflective scene reconstruction, see [IKL+10].

2.4.1 Flame-Sheet Decomposition

The idea of using sheet like structures to represent flames was first introduced by Hasinoff in 2002 [Has02], then refined in 2003 [HK03] and again in 2007 [HK07]. The earlier work introduces an image formation model with density based on self-emission which is used in all the following literature in this section. The model assumes negligible atmospheric scattering, which should hold provided there is not substantial smoke in the input images. This is a reasonable assumption, since almost all literature treats smoke and fire as separate phenomena to be modelled. A collection of three-dimensional Gaussian blobs with varying standard deviations are optimised to fit the images, however the results suffer from over-fitting.

The subsequent work uses sheet-like structures instead of Gaussian blobs to produce a better result. Inspired by fake candles that use lit sheets of cloth, the authors introduce these 'flame-sheets' (earlier work) or 'density-sheets' (later) as spatially compact solutions to the two-view reconstruction problem. A density-sheet minimises the spread of den-

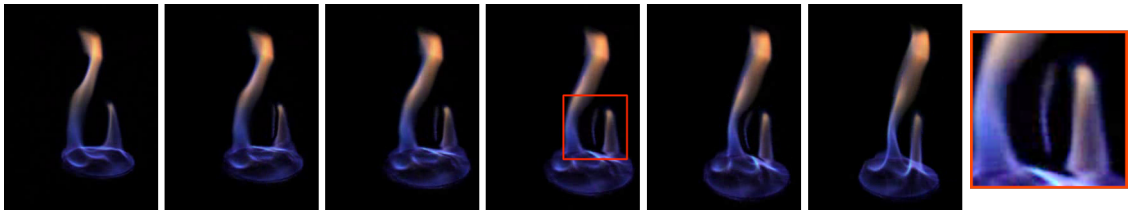


Figure 2-11: The flame-sheet method applied to the burner data set (Images from author's website[1], emphasis added)

sity to a connected two-dimensional sheet embedded in three dimensions. The concept is illustrated in Figure 2-10. Each pair of views generates two possible density sheets which are photo consistent, and by using more views a basis of sheets is formed. This construction means that view interpolation is possible in the same axis of rotation as the cameras (around the flame), however the reconstructed flames will look 'sheet-like' when viewed from an angle above or below the original. Furthermore, while the approach gives good results for simple flames, it produces artefacts when applied to more complex flame structures, essentially from treating the flame as a two-dimensional phenomenon. Notice in Figure 2-11 that the reproduction looks realistic at both extremes, which correspond to two cameras out of the eight camera set up, however part of the smaller flame is simply floating in mid air in the in-between angles.

2.4.2 Image-based Tomographic Reconstruction of Flames

Ihrke and Magnor's approach [IM04] is a true three-dimensional approach, being a form of computerised tomography. This work is pre-eminent in the field, and is used as ground truth in the later work by Hasinoff. The method is described in detail here.

As in [HK03], this method assumes that the fire can be modelled as a three-dimensional density field ϕ of fire reaction products

$$I_p = \int_c \phi \, \mathrm{d}s \tag{2.4.1}$$

where ϕ is the three-dimensional fire density field, I_p is pixel p's intensity, and c is a curve through three-dimensional space, in particular the back-projected ray of pixel p.

In order to invert Equation 2.4.1, it is assumed that ϕ can be represented as a linear

[1] `http://people.csail.mit.edu/hasinoff/fire/`

combination of weighted basis functions ϕ_i as such:

$$I_p = \int_c \left(\sum_i a_i \phi_i \right) \mathrm{d}s \tag{2.4.2}$$

Now the sum and coefficients a_i can be moved out of the integral:

$$I_p = \sum_i a_i \left(\int_c \phi_i \, \mathrm{d}s \right) \tag{2.4.3}$$

Notice that this represents a linear system over the basis functions.

$$\mathbf{p} = S\mathbf{a} \tag{2.4.4}$$

Where $\mathbf{p}$ is a vector containing intensities at each pixel, S represents the contribution of each back-projected ray from each pixel over each basis function, and $\mathbf{a}$ is a vector containing the coefficients for those basis functions. In other words, a single row of the linear system says that pixel p_i is equal to a row of S, which gives the contribution of each basis function to this pixel, multiplied by $\mathbf{a}$, the global value of each basis function. It is possible to reconstruct the entire density field, given the basis functions and their coefficients from $\mathbf{a}$.

Solving this linear system depends on the choice of basis function. To make computation efficient and simple, the box basis function is an attractive choice.

$$\phi_i^{\text{box}}(x, y, z) = \begin{cases} & x_{\min}^i < x \leq x_{\max}^i \\ 1 & y_{\min}^i < y \leq y_{\max}^i \\ & z_{\min}^i < z \leq z_{\max}^i \\ 0 & \text{else} \end{cases} \tag{2.4.5}$$

This has local support, so the matrix S will be sparse. Furthermore, axis-aligned basis functions can be chosen, which will mean computing their contribution to each back-projected ray is of $\mathcal{O}(3n)$ rather than $\mathcal{O}(n^3)$. In fact, for pure simplicity a one to one mapping between basis functions and voxels can be used, for the desired resolution.

Using the box basis function, the elements of S can be found based on the camera calibration. A full proof is in the paper, however it stands to reason that the elements of S

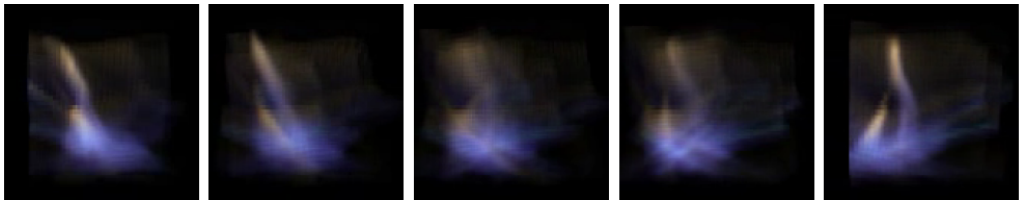


Figure 2-12: Non-restricted solution to the linear system

are given by

$$S_{ni} = \|x_2 - x_1\| \tag{2.4.6}$$

where x_1 and x_2 are the intersections of the back-projected ray from pixel i with basis function (voxel) n. This makes sense – a voxel which is directly in the path of a pixel's ray should provide a higher contribution than if the ray just goes through the corner, like hitting the edge of an object. Solving the system gives the weights of each basis function $\mathbf{a}$, in other words the density in each voxel.

To solve the system, a least squares solution to 2.4.4 is solved:

$$\mathbf{a} = (S^\mathsf{T} S)^{-1} S^\mathsf{T} \mathbf{p} \tag{2.4.7}$$

The size of S makes direct computation infeasible, since $S^\mathsf{T} S$ will be the size of the number of basis functions squared. To avoid having to compute the term directly, an iterative method is necessary. A modified version of Conjugate Gradient Least Squares [Han98] method is chosen for this task. The details of the iterative process are in the paper, however it is modified to project the solution into a non-negative subspace at each iteration by setting all negative entries to zero. The L-curve termination criterion [Bjö96, Han98] is used to find a trade off between a smooth result and accuracy of fit. This is the solution that maximises the curvature of the line given by $\frac{\|\mathbf{x}_k\|}{\|S\mathbf{x}_k - \mathbf{p}\|}$, the quotient of the norm of the solution over the norm of the residual.

When solving this system, the results show that the number of views is not enough to restrict the solution to a clean result. Notice the ghosting artefacts in Figure 2-12. The solution for this is to compute what the authors call the 'visual hull restricted solution'. This is to calculate the visual hull of the original images [Lau94] – in other words projecting their silhouettes and taking the intersection – and then remove the columns corresponding

Figure 2-13: Visual hull restricted solution to the linear system

to any basis functions that lie outside of this intersection from the system before it is solved. Then, the values of a_i are replaced as zeros. This is in effect forcing the densities to lie within the silhouette of the cameras, which gives a much cleaner result, as can be seen in Figure 2-13.

The process is repeated for each frame and each colour channel of the video being reconstructed. Note that the construction of matrix S only depends on the camera setup and the basis functions. Provided it is generated fully, and the visual hull restriction is done temporarily for each frame, the largest computational cost only need happen once.

This image-based tomographic method can be considered ground truth for the input images that are provided. Although one minor adjustment is made in a later paper, the core idea remains the same and for a high enough voxel resolution the results would be the same.

The model does not make any considerations for differing brightness or for colour correction between cameras. This could lead to some inaccuracy, especially when there is noticeable variation in the data; but the results do not seem to suffer overly for it. The authors use flames filmed from eight cameras and claim that results produced with four or more views look sufficiently accurate when animated. The technique does not produce sensible results below this number, one disadvantage compared to flame-sheets (Section 2.4.1) which is designed to work with two.

This technique is easily the best method of producing three-dimensional flames from video. However, like the others presented in this chapter, the results are not editable. Voxel models are produced for each frame of the input video, however they cannot be edited other than by changing values directly. One input video results in exactly one output, and if the flame does not look as intended, there is no choice but to refilm it.

Adaptive Grid Optical Tomography

In a later paper [IM06], the authors adapt the structure of the box basis functions so that there is a higher resolution in more complicated areas of the flame. This is done via an iterative, adaptive octree structure. The reconstruction is performed as above for a given set of box basis functions. The result is projected back onto the image plane, and the basis functions which contribute to the greatest error are split into eight smaller basis functions, in an octree structure. This is done by augmenting the matrix S. The authors use this to increase the resolution of their reconstructions. In the previous work, the input images were shrunk, and 64 voxels per dimension was used for the reconstruction. In the later work, they achieved 128 voxels per dimension in the most complicated areas, and the full 320x240 images.

While a more efficient structure is obviously a useful feature, the adaptive grid requires recalculating the matrix S for each input image, depending on which areas are most complicated. Given the time since the previous technique was published, computer hardware has become significantly more powerful. This is elaborated in the next chapter, suffice it to say that the resolution of the more efficient method was easily achieved with the original on modern hardware. Therefore given the saving in computation by using a single S matrix for each image, both techniques are worth considering.

2.5 Conclusion

This chapter presents a few findings about the current literature of flame modelling. The first and foremost, is that *model acquisition is difficult and expensive.* No matter the technique used to animate an artist's created flame, or represent one from video data, none of the methods currently available focus on simple control. Methods from computer graphics produce highly detailed results, however require complicated model production with unintuitive parameters. Finding models from video is a simple solution to this, however, these techniques do not allow control over the result. A method that can be controlled by both video and a user is an open problem in the field.

Furthermore, three-dimensional video-based methods focus on animating or reconstructing specific flames. It would simplify the model acquisition process considerably if a single source flame could be used to generate more examples of a similar type. Especially if there were intuitive controls associated with this as well.

Finally, no methods in the current literature for flame reconstruction consider a single camera viewpoint, and some state it is not feasible to do so.

The work presented in the rest of this document will address all of these shortcomings with novel solutions.

Chapter 3
Capture, Details, and Sources of Flame Videos

The contribution from this chapter is a dataset of flame videos, along with their three-dimensional reconstructions. Also valuable is the implementation of Ihrke and Magnor's Tomographic method for acquiring these models from video. Together these form the start of a library, currently of flames, but that could be expanded to include any transparent phenomena. It also discusses the practical and technical details about the data that is used throughout the thesis such as the format and appearance of flames, which while not a contribution in itself, is essential to properly comprehending the results from rest of the document.

3.1 Overview

When designing imaging systems – be they cameras, monitors, or something else – out of necessity the human visual system is used as the golden standard. While a user might subvert this for artistic reasons, it is usually expected that taking a photo of a scene should appear on a monitor as close as possible to the way the scene appears in real life. However, the human visual system is complex and inconsistent, and the devices that capture and display the images have many other limitations that one must work around when trying to use them to capture for scientific reasons rather than artistic. These imperfections affect the input and output of the main contributions of this document, and so are discussed first.

Following this are the details of the data that was used in this project: the pre-existing

data, and the data that was captured specifically for this purpose. The process of capturing the new data is detailed, as is the specific implementation for converting the videos into three-dimensional reconstructions. Technical limitations with the data are also discussed, some that were addressed with the new recording and some that remain an area for future work.

3.2 Technical and Perceptual Effects and Limitations

The area of visual perception research, including how it relates to computer vision and graphics, is large and active. It is not the subject of this document, however some basic information will help frame many of the results. A broad overview covering many fields of visual science is given in [Pal99].

Flames differ from many typical subjects of capture because they emit, rather than reflect, the light that makes them visible. This makes capture easier in some ways, more difficult in others, when compared to non-light emitting phenomena. For a start, consider the comparison made in the previous section, of using the human visual system as a golden standard. Humans are particularly good at adjusting to different levels of brightness. Our pupils change size to accommodate brighter or darker scenes. When we look at a scene with a lot of dynamic range, we can refocus on different areas, and change the 'aperture' of our eyes (to use the camera term). This builds up a mental picture of a scene with much more range than a camera can capture.

When we view a scene featuring a flame, the flame is usually many times more bright than the other elements. Quantitatively measuring brightness values is highly dependent on context, however doing so comparatively can give a sensible idea. Using the standard light-meter in a DSLR camera, the centre of a candle flame was measured as over 7 f-stops brighter than a sheet of white paper under the same indoor lighting. One f-stop of brightness is a factor of two brighter, so this is a difference of 128 times. This difference will be drastically reduced in a bright outdoor environment, because the ambient light will impact the paper, but not the self-emitting light of the candle. However, it is safe to say that when we view a flame, our eyes are adjusting to correctly expose the ambient surroundings, not the flame. This leads to most flames losing any definition of colour in their brightest areas, sometimes appearing entirely white, not from temperature but from contextual brightness. This gives us the impression of what we expect a flame to look like.

The flame itself also has a range of brightness. Looking carefully at a candle flame, there is a faint blue glow around the base, followed by a much brighter yellow flame above. The blue flame is being produced by a clean, fully oxidised reaction, whereas in the rest of the flame there is not enough oxygen present, and this is producing glowing soot particles (see Section 2.1). Our eyes are sensitive enough to detect the blue light, despite it being many times dimmer than the yellow. Using an 8-bit camera, it was not possible to find an exposure setting that made the blue section visible without completely losing any detail in the yellow section. Higher bit depth/dynamic range cameras are an interesting area for future developments.

The typical overexposed look of a flame actually represents a loss of information. Taking a perfectly exposed flame, it is possible to then manipulate the image later to 'blow out' the brighter areas and form an image with white areas more typically seen, but it is not possible to retrieve information later if an area is overexposed. This applies to underexposed (black) areas as well, and as mentioned it was not possible to capture the entire range of detail in the captured images. Since the purpose of the images was to create reconstructions, it was decided that it was better to lose dark areas of the flame than to have overexposed areas. Zeroes in the images would mean this area just wasn't visible in the reconstruction. A capped value however would invalidate the image formation model: that the sum of the true densities equals the pixel value. In the overexposed case, the sum of the densities is actually higher than the pixel value.

Another complication is the medium we use to view images of flames. These will either be on a monitor, or printed in some format. The former method is entirely self-illuminated, the later method is not. Both of these, therefore, lose a significant amount of contrast compared to viewing a flame in person.

For these reasons, the flames in this document may look less bright than one would expect. The brightness of the flame is not being measured in any objective way: it is impossible to do this meaningfully, because the eye and the camera must see the flame differently out of necessity; similarly the viewing medium and conditions render any idea of objectivity moot. In some cases the brightness of images in this document have been increased to enhance visibility. In general, flames will not appear in the typical overexposed way, unless they are from another source. Whenever several images are being compared, they will be treated in the same way.

One benefit to filming light emitting phenomena is that they can easily be isolated against a black background by using a dark room and a low exposure. This is convenient with

the decision to tend towards a darker exposure anyway. For this reason and for greatest contrast, the flames in this document will be presented against a black background. The three-dimensional transparent models can easily be placed into any environment if required; making the two look realistic in context is beyond the scope of this work.

3.3 Three-dimensional Flame Data

In this section the two sources of three-dimensional flame data are described. These are the only currently available sources of flames that are from real video data, and produced by a fully three-dimensional process. In their original format, they are synchronised videos of flames from multiple camera angles. These are converted into three-dimensional models using an implementation of Ihrke and Magnor's flame tomography method [IM04], discussed in the next section.

3.3.1 Pre-existing Data

The only example of pre-existing data that meets the criteria is from the same project that produced the method for converting the images into three-dimensional models [IM04]. This video is of a burning dish of alcohol, it has a resolution of 320×240 pixels, bit depth of 8-bit, and was filmed at 15 frames per second for approximately 300 usable frames total, with 8 cameras simultaneously. In the first half of the video, the flame is left alone, in the second half it is being blown from one side.

The data is very old as of the time of writing, and unfortunately has many technical limitations. Two had particularly negative impact: first, the frame rate of 15 frames per second. Flames move quite erratically, especially more complicated flames. The difference between two consecutive flames in the original video was very large, which meant that it was hard to ascertain exactly how the flame was moving between consecutive frames. Second, many of the frames in the original video suffer from some visual problems, most commonly motion blur, although there were problems with sensor noise as well. While motion blur can be visually pleasing in a video, in this case it meant that the three-dimensional reconstructions were blurred also. While the techniques that will be presented in Chapter 5 are relatively robust to noisy data, in many frames there simply wasn't a flame that was clearly defined enough to create a clean result. These two problems compound,

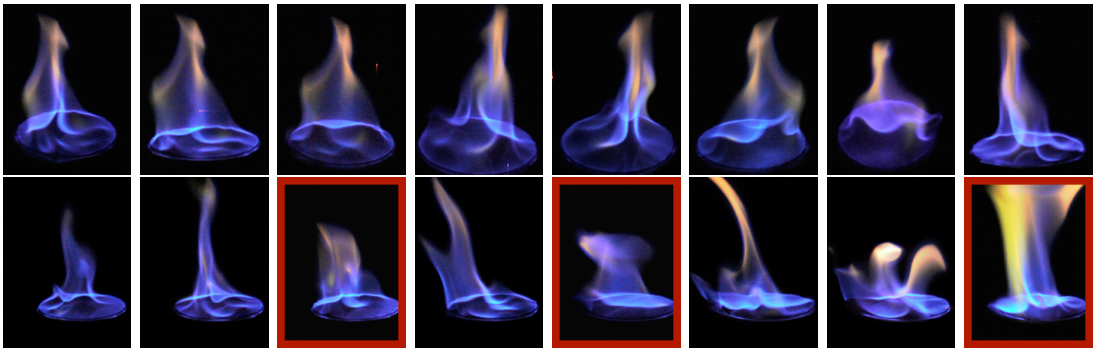


Figure 3-1: Row 1: eight views of a single flame
Row 2: eight other frames from the same view
Note motion blur or noise in several frames, severe cases highlighted

because in many cases having no usable data from a particular flame meant that motion was even harder to analyse in the adjacent ones.

Figure 3-1 shows the alcohol data set, both clear frames and frames that suffer from obstructive technical issues.

There was only one other known source of multi-view synchronised flame capture, which is the torch data set from [HK07]. However, this data is only in stereo, therefore isn't sufficient to create a full three-dimensional reconstruction.

3.3.2 Filmed Data

One of the contributions of this project is a new dataset of flame videos that can be used to create high quality three-dimensional reconstructions. At present this contains two videos, one of a candle flame which is just over 6000 frames long, and the other is of a lighter which is just over 1000 frames long. Like the alcohol video, the candle is blown from the side for the second half of the video. Both videos have a resolution of 2048×1088, bit depth of 8-bit, and were filmed at 100 frames per second, with 6 cameras simultaneously. Calibration to find the camera parameters was done using [SMP05] and a custom built circuit powering a single green LED.

An earlier attempt at filming a candle using DSLR cameras was made, however the inability to synchronise precisely meant that the multi-camera calibration gave too high of an error to produce meaningful results. Interpolating between frames was considered, however since the data was going to be used as ground truth it was decided that only true

Figure 3-2: Six cameras around an unlit candle, the set-up used to capture the data

flames should be used.

The new videos are state of the art of the currently available data. Their creation also establishes a known pipeline for capturing further types of flame, or other transparent phenomena, using the same equipment.

There are two limitations which were caused by available hardware, specifically the lenses. First of all the focal length is very wide. This means the cameras had to be very close to the flames, and even still they did not fill up the frame at the closest focus distance. Second, they have a stepless aperture ring. While care was taken to set each aperture to the same value, some variation will inevitably exist, causing a difference in brightness between the images. This was not found to be a problem in reconstructions, however, a stepped aperture ring would cut down on errors. A picture of the camera set-up, with an unlit candle, is shown in Figure 3-2.

Some frames from the filmed data are given in Figure 3-3. The images presented have been cropped from their full size to just show the flame. This process was centred around the flame, so any translation movement of the burning material, particularly the handheld lighter, is not necessarily reflected. The actual reconstructions do reflect this motion.

3.4 Flame Tomography

An implementation of Ihrke and Magnor's flame tomography method [IM04] described in Section 2.4.2 was produced in Matlab. It calculates the intersection between each back-

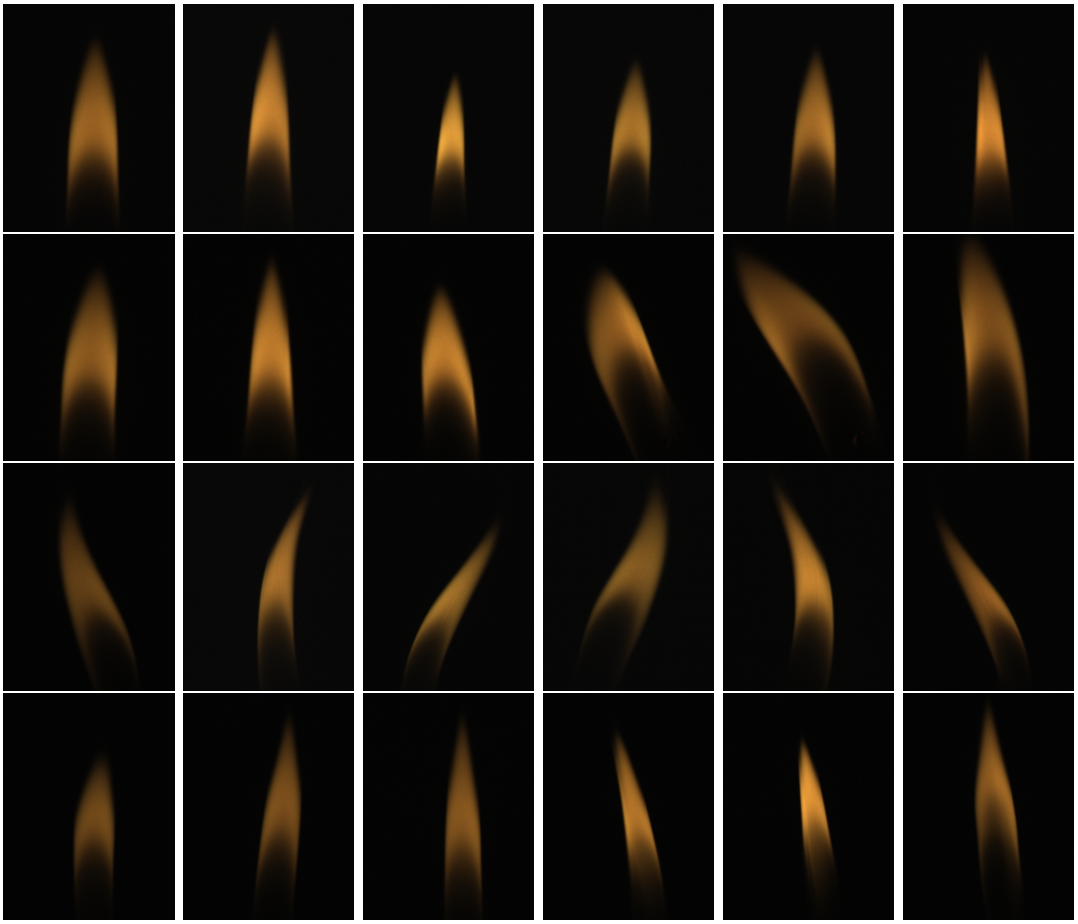


Figure 3-3: From top to bottom: six views of a single candle frame, six other frames from the same view, then same again for a lighter

projected pixel ray and each of the $3(n+1)$ planes for the voxel grid, then sorts the intersection points in order of distance along the ray, and then computes the intersection distance for the corresponding intersected voxel (this is the value for S_{ij} in the original method). This is a very simple voxel ray casting method that produces intersection points, others may be more efficient. The matrix S is very large, so in order to save on memory, a sparse representation is formed. Finding an unknown number of intersections results in usually allocating more space than is necessary; during construction of the S matrix, the implementation unloads intersection results for each image to disk temporarily, then loads them at the end to form the sparse matrix for all images. In the original authors' implementation a resolution of 64 voxels per dimension was used. Using this memory-saving technique and on much newer hardware, it was possible to create reconstructions

with over 200 voxels per dimension, with much higher resolution images.

To create the visual hull, either Otsu's thresholding method [Ots75], or the triangle method described in [ZRL77] was used, depending on which produced better results for the source flame. The authors use the L-curve criterion to determine the termination of the iterative least squares process, however in tests it was found that over-fitting artefacts were still present in many results. Smoothing with a Gaussian kernel greatly improves the appearance.

One of the advantages of this algorithm (over the adaptive grid method [IM06] for example) is that the matrix S only needs to be computed once for the entire video. This process is still costly, and the implementation takes about 8 hours on modern hardware for a high voxel and image resolution. The candle dataset was reconstructed at 256 voxels per dimension, with an image resolution of 1024x544. These parameters were chosen to maximise the usage of memory, not speed. The implementation can run in parallel up to the number of cameras, which significantly increases speed but means that the memory-saving technique can not be used, so the resolution has to be lower. Once S is computed, the remaining operations take around a minute, the mostly costly of which is solving the linear system. Note that since this is done for each colour channel, reconstructing a 6000 frame sequence is still an extremely long computation time (in the region of 12 days on one machine). A focus on optimisation could improve the speed of both sections of this implementation.

The results of the three-dimensional reconstructions are shown in Figure 3-4. Note that these images were not projected into the exact same camera specifications, they were rendered using a simple voxel renderer, like all of the flame images in this document. The viewpoints should be close to those used in Figures 3-1 and 3-3, however they are not exact.

3.5 Conclusion

This chapter forms a platform for the rest of this document. The hypothesis, from Chapter 1, was: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* The decision to use real video to drive editable models necessitates good quality real video to design for and test methods on. Since the models should be three-dimensional, it is highly beneficial to have three-dimensional models from real video.

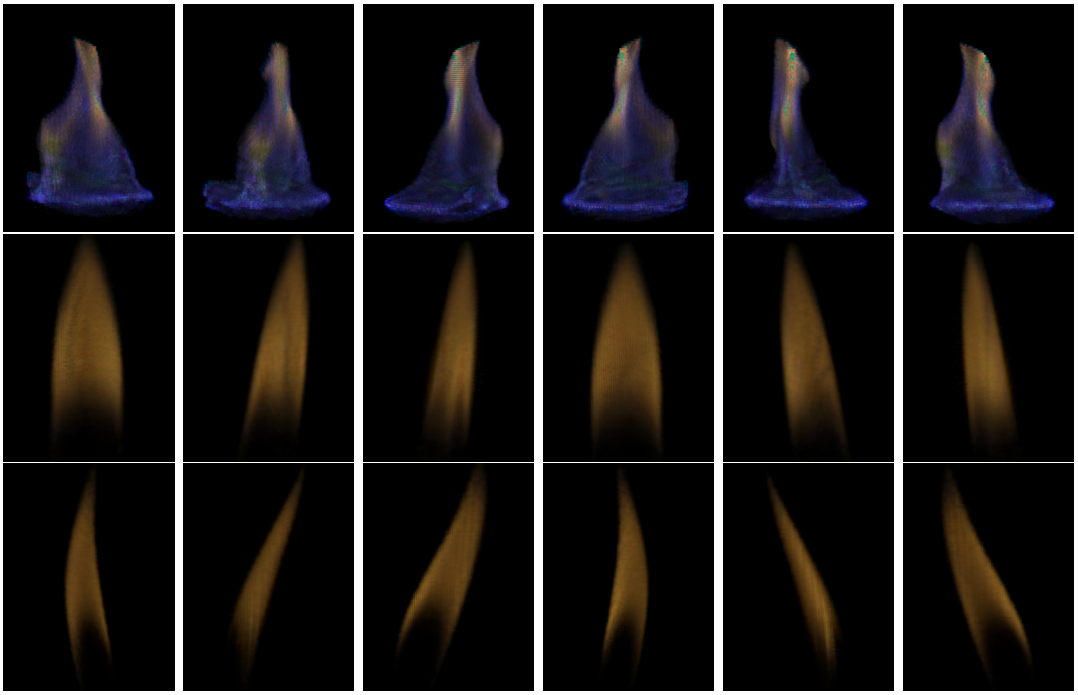


Figure 3-4: Multiple views for three reconstructed flames, one from the alcohol video (two views omitted), one from the candle, and one from the lighter

This chapter presented two new sources of this data which are state of the art for the field. However, this sort of voxel model is not editable. Subsequent chapters will show that editable models can be created based on these models, and these can generate brand new animations which look similar to the original flame video.

This chapter also covered more technical details and limitations about the images in this document, which will be assumed subsequently.

Chapter 4
The Flame Core Model

This chapter presents the first main contribution of the thesis, which is an intuitively editable model for flames. This is called the "flame core model". In this chapter the model is introduced and then described in mathematical detail, plus a simple rendering algorithm is given. It is shown how a flame in this format is editable in a way that voxel methods cannot achieve. It does this while being more efficient in terms of space and has the appearance of a realistic flame, with the trade-off of requiring an additional step to render.

4.1 Overview

The concept of the flame core is based on the observation that a lick of flame appears to be radially symmetric around a central 'core'. This defines the overall direction of the flame, and other parameters control the shape and brightness of the density that surrounds it. In some flames, such as those of a candle, this concept has a physical parallel; in this case the wick is the core. There is also precedent for this observation in the flame animation literature; it inspired the work Structural Modelling of Flames (see Section 2.2.5). Some flames consist of several distinct licks, and each of these would have a separate core, which would then combine together when rendered. In this section 'flame' will usually refer to a simple flame with just one lick, such as that of a candle. The concepts transfer into more complicated flames by combining multiple smaller licks.

In the flame core model, the centre of the flame is defined as a simple line in three-dimensional space. This is called the core, and it controls the overall shape of the flame: a core that is bending to the left will result in a flame that will do the same. This concept

Figure 4-1: Illustration of flames with central cores (in black) that define their shapes

is illustrated in Figure 4-1.

This core is one of two elements that make up a flame. The other is a density field that is wrapped around the core. Since the aim is for the model to be intuitively editable, a set of simple parameters is used to describe how the appearance changes along the core of the flame, rather than requiring an explicit density field. The density is assumed to be radially symmetric around the core. This greatly simplifies the parameters needed to represent and therefore edit the flame, while still giving a flame-like appearance.

This flame core model is the first main contribution of the thesis. It is widely descriptive, easily and intuitively edited, and as subsequent chapters will show, can be easily fitted to real world data.

4.2 Justification

As previously mentioned, the flame core model has two parts: the core which defines its overall shape, and the density function which defines the density surrounding the core. The core is a single line in three-dimensional space. Complicated flames that seem to require a more complex, possibly branching structure are handled by allowing multiple cores, each with corresponding density parameters, within the same flame.

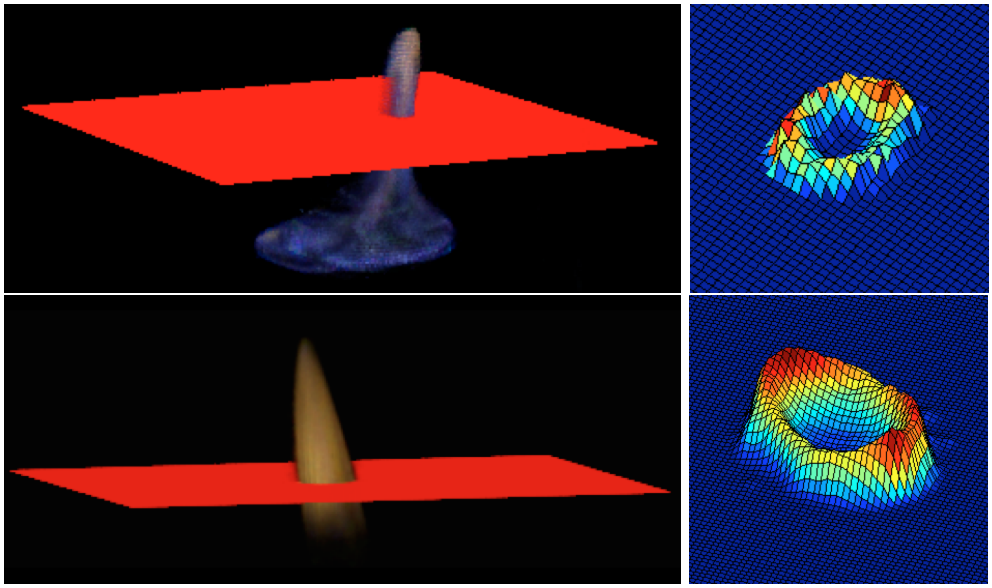


Figure 4-2: Horizontal slices of density through a flames from video

The function that defines this density in the flame core model is inspired by probability distributions, and based on both the science of flames and observations in real data. As described in Section 2.1, there are two components contributing to the brightness in a flame: light that is produced by the reaction between the heat and fuel of the flame and the oxygen in the air, and light that is coming from glowing soot particles dispersed throughout the volume. How cleanly the fuel burns will determine which component produces more light. However, in either case, the majority is coming from the edge of the volume. This is because the heat of the flame is also being produced at the edge, and so this is where the soot particles are brightest. Provided there is some soot, there will be some light emitting from the centre, but always less than the edge.

This observation is reflected in the data: the fall-off of the brightness at the edge of the flame is a Gaussian bell-curve like shape. However, rather than falling to zero at the centre, there is some brightness present throughout the flame. Figure 4-2 shows cuts through real flames from video, showing a circular slice with the majority of brightness falling around the edge of the flame, and with lower but non-zero density at the centre of the flame. Analysing this density in terms of distance from the centre gives the same result, shown in Figure 4-3.

Early attempts at finding a model included using a Gaussian mixture model, since representing the distribution of density itself was a goal. This led to overfitting, which is

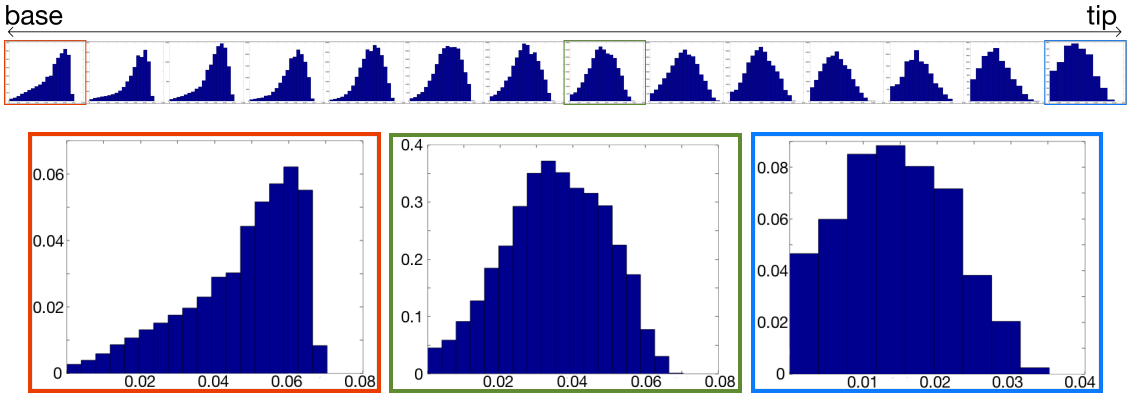


Figure 4-3: Histograms of density at various points along a flame captured from video Graph shows distance from centre vs. density, compare to the planar version of a single slice in Figure 4-2

a result also described in [Has02]. Because the light-producing reaction is happening at the edge of the flame, the Gaussian distribution is a poor fit to the density directly. On the other hand, a second derivative Gaussian produces a better shape – it is bimodal which could account for the densities at either edge of a lick. A mixture of multivariate second derivative Gaussians was considered, however it was dismissed in favour of a more direct structural approach. It was decided that this would be easier to work with, and the structural element itself would prove useful to the editability.

Instead, a function was developed to model the density in terms of distance from the centre. This was designed to model the shape closely, while having few parameters that correspond intuitively to the shape of the density and thus the visual quality. This is vital for the editability of the flame, the key objective of the work. There are three components in total. Two correspond to the two sources of light: one at the edge of the flame, and one dispersed throughout the flame. The third component controls the distance from the peak of the density to the edge of the flame, allowing a soft or harsh edge (evidence is seen for these also in Figure 4-3).

Firstly, consider the dispersed component. The density at the centre of the flame is non-zero because heat causes soot particles to glow. In relation to the light at the edge of the flame, this contribution is quite low, and so the specific form is not as important; the main requirement is that the density is non-zero in the centre to provide the correct contrast. Therefore, the component that accounts for this light was chosen to be a simple constant. This is similar to an ‘ambient light’ term used in many graphics applications.

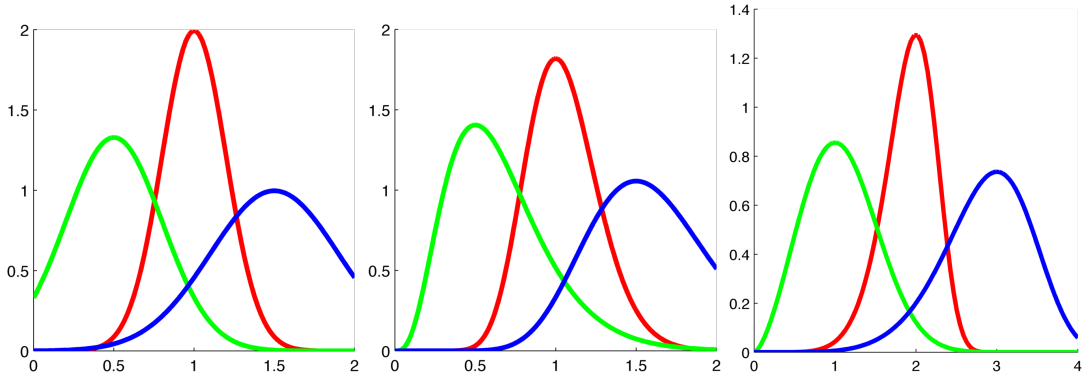


Figure 4-4: Examples of the normal, gamma, and Weibull distributions (left to right) Note the data skew is normal, positive, and negative, respectively

The component which allows the sharp drop-off at the edge is a simple step function. It has a single parameter specifying the location of the edge of the volume, above which the density is assumed to be zero. Again this is a simplification, however assuming that the main light component has a smooth tail, this step function is an easy way to give a variable fuzziness.

This leaves the component which controls the Gaussian-like shape at the edge of the flame. There are multiple choices for this function. The obvious choices are inspired by probability distribution functions, with the normalising constant replaced with a scale factor to control brightness[1]. In particular, the probability density functions of the normal, gamma, and Weibull distributions were considered. All three distributions are controlled by two parameters, and give the necessary bell curve. In experiments it was found that the Weibull distribution was the best fit for the shape of real data. This is possibly due to its asymmetry: it has a negative skew, i.e. a shorter tail – the decline from its highest value to zero – than the other two distributions. However, the parameters do not intuitively correspond to the shape of the curve, which conflicts with the main objective of this work. In particular, in Section 1.3.1 it was stated that non-orthogonal parameters are not intuitive. In order to make the flame look brighter with the Weibull function, one might increase the scale parameter; however, this also has the unintended consequence of moving the edge of the flame. This problem exists for the gamma distribution function as well.

The parameters claimed here to be most intuitive for editing this component of the density are: the brightness at the peak, the position of the edge of the flame, and the width of the

[1]In the case of a normal distribution, this is sometimes called a Gaussian function.

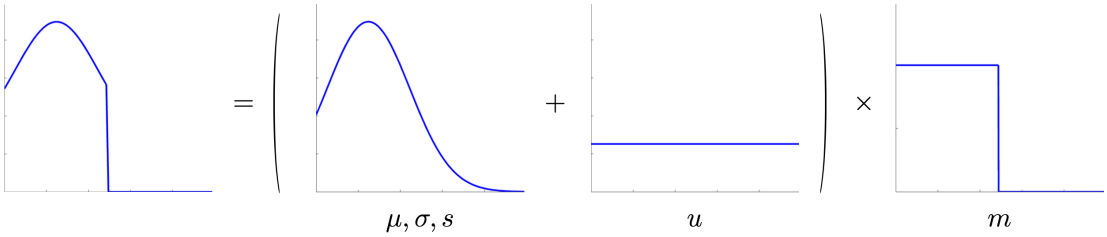


Figure 4-5: Three components and five parameters that make up the density function

fall-off from the peak – the fuzziness of the flame. These correspond to the maximum value, the mode, and the 'variance' respectively. The normal distribution function's parameters coincide perfectly with this intuitive control. It is possible to reformulate the gamma distribution function to achieve this, which is discussed in Appendix A. However, the normal distribution gave a better fit to real data than the gamma distribution. Despite the fact that it does not fit as well as the Weibull distribution, the immediately intuitively editable parameters greatly simplify the model, and the difference in visual appearance is not significant. For these reasons, the normal distribution was used.

In summary, the density function for a given point along the flame is a composition of three components. One: the density function from a normal distribution with the usual normalisation constant replaced by a scale parameter. Two: a uniform contribution added to the whole density. And three: a step function that sets the result to zero above a certain distance – the very edge of the flame. These components are illustrated in Figure 4-5. In the next section, these components are defined formally.

4.3 Definition

In this section the flame core model will be defined in detail.

As previously mentioned, a flame is modelled with two parts: a line in $[0, 1]^3$ giving the flame's core, and parameters controlling the appearance of the density. These are given by functions with a domain of $[0, 1]$, where the result of the function at 0 represents the parameter at the start of the core, and 1 represents the end. The flame core model is designed to be used in colour, since this is very characteristic of flames. This is achieved via separate density functions for each colour channel (red, green, blue), although some of the parameters are shared. For simplicity here, the details are given as if the flame were

greyscale; the small modification to create a colour flame will be detailed later.

Formally, a flame is modelled by a tuple, (c, d). For $h \in [0, 1]$ representing a position along the core,

$$c : h \mapsto (x, y, z) \in [0, 1]^3 \tag{4.3.1}$$

gives the point along the core in three dimensions at position h. Hence the function c defines the full shape of the core.

The density at the same point on the core is given by d.

$$d : h \mapsto (\mu, \sigma, u, m, s) \in \mathbb{R}^5,\ \sigma \neq 0 \tag{4.3.2}$$

These five parameters are used in the density function to specify brightness in terms of distance from the core x, as described in the previous section and illustrated in Figure 4-5. This is given by

$$\phi(x, \mu, \sigma, u, m, s) = \begin{cases} s\left((u-1)\exp\left(-\left(\frac{x-\mu}{2\sigma}\right)^2\right) + u\right) & \text{if } x \leqslant m \\ 0 & \text{otherwise} \end{cases} \tag{4.3.3}$$

Hence, to describe each parameter visually

- μ gives the location of the peak of the density;
- σ gives the spread of the density;
- u gives the contribution of uniform density, the ambient term;
- m gives the edge of the flame; and
- s gives the overall brightness scale factor.

Figure 4-6 shows an illustration of the relationship between the core and density functions. The density is illustrated as a line shown in red. It shows the brightness for a horizontal slice of the flame, as if it were being plotted in two dimensions, distance versus brightness.

Note that the parameters should be considered restricted to within a range that creates a sensible looking flame. At the minimum, m, u and s should be confined to $[0, 1]$, and σ to $(0, 1]$. Interestingly, a negative value for μ is sometimes desirable, if the density should be entirely decreasing, which sometimes occurs at the top of a flame. $-m$ can be used as a sensible minimum for sanity checking μ, and its maximum is obviously 1 also.

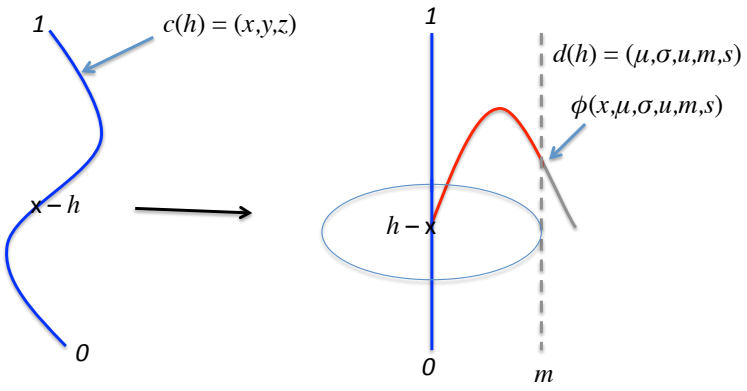


Figure 4-6: An illustration of the flame core model

4.3.1 Colour Flames

The above gives a formal definition for a greyscale flame. As mentioned, a colour flame can be represented by three density functions, one for each colour channel. In fact, it was found to be beneficial for just two parameters to vary between the three functions: s and u. These give the colour value, and the contrast within each channel, respectively.

A colour flame can be formally denoted (c, d_r, d_g, d_b) with the definition of d unchanged, or as before with $d : h \mapsto (\mu, \sigma, \mathbf{u}, m, \mathbf{s})$ where $\mathbf{u} = (u_r, u_g, u_b)$ and $\mathbf{s} = (s_r, s_g, s_b)$.

4.4 Rendering

A simple algorithm to render a flame core model is given here. The rendering produces a voxel representation, which can then be rendered further using a traditional voxel rendering algorithm, introduced in [Lev88, DCH88]. Many recent algorithms exist which give different levels of quality, rendering time, and interactivity. Images in this document of three-dimensional volumes are rendered using a texture mapping technique, based on the concept introduced in [HS89b], which gives real time interactivity on modern hardware. More sophisticated ray casting techniques may produce better quality results, a recent survey of state of the art approaches utilising GPU acceleration can be found in [BRGIG+14].

A rendering of a flame with multiple cores is equivalent to rendering the cores individually and summing the contributions from each. Hence, what follows is the method for rendering

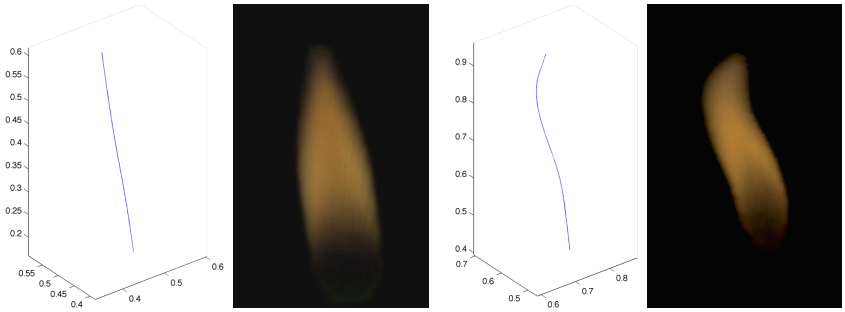


Figure 4-7: Two simple flames with their cores plotted as lines

a single core.

For a rendering with a resolution of N voxels in each dimension, the centres of the voxels scaled to lie in $[0, 1]$ are:

$$\mathbf{v}_{ijk} = \left(\frac{i}{N} + 0.5, \frac{j}{N} + 0.5, \frac{k}{N} + 0.5 \right) \text{ for } i, j, k = 0 \ldots (N-1) \tag{4.4.1}$$

For a given voxel (x, y, z) with centre point $\mathbf{v}_{xyz}$, find:

$$h = \underset{x \in [0,1]}{\operatorname{argmin}} \| c(x) - \mathbf{v}_{xyz} \| \tag{4.4.2}$$

Let

$$\delta = \| c(h) - \mathbf{v}_{xyz} \| \tag{4.4.3}$$

be the distance of this voxel from the core.

The density parameters for this point on the core are given by $d(h) = (\mu, \sigma, u, m, s)$. Then the density in voxel (x, y, z) is simply given by $\phi(\delta, \mu, \sigma, u, m, s)$ as in Equation 4.3.3.

Examples of two very simple flames with their corresponding cores are given in Figure 4-7.

Rendering was implemented by following this process for each voxel at the desired resolution. This has $\mathcal{O}(n^3)$ complexity, which will mean a long running time for high resolution renderings. An optimisation was made which moves along the core, creating circles of points that are perpendicular to the core, with a distance equal to the maximum at this point (given by the m parameter). A bounding box is found from these points, which is used to prune the voxel points, setting any values outside of the bounding box to zero. The three-dimensional renderings in Figure 4-7, which use a medium resolution of 160

voxels per dimension, take around 4 seconds using this optimised method, not including the voxel renderer.

4.4.1 Preventing a round base

In a normal image of a flame, the base of the flame is often obscured by the object that is alight. A flame core rendering can easily be inserted into a scene with another object to achieve this effect. However when being displayed alone, it creates an odd, seemingly unrealistic effect if the flame has a hemispherical base. Therefore, renderings in this document modify the base of the flame to produce a flat result, which looks more visually appealing.

Normally the base is round because the voxels directly below the start of the core $c(0)$ are within the range to be rendered in the same way a voxel on the same horizontal plane is. To avoid this, the core is extrapolated to create a point just below the first point:

$$\mathbf{p} = c(0) - n(c(h) - c(0)) \tag{4.4.4}$$

Where $h > 0$ is a value very close to 0 (the next point on the core, if it is represented in a discrete fashion), and n is also a small scalar, $n = 0.1$ for example. This new point $\mathbf{p}$ is appended as the new start of the core for the purposes of finding the nearest points only. Then, any voxels which match to this point are set to 0, and every other voxel is treated normally.

This procedure is not required for the top of the core. This is for two reasons, firstly most flames taper out into a point, so the round shape is small enough that it is not noticeable. Secondly there are some flames which do actually have round tops, in which case the result is obviously desirable.

4.5 Control

In Chapter 1 it was stated that an objective of this work was that the model should be intuitively editable by users, meaning that parameters should be directly linked to visual appearance. In this section the control and editability afforded by this model is demonstrated.

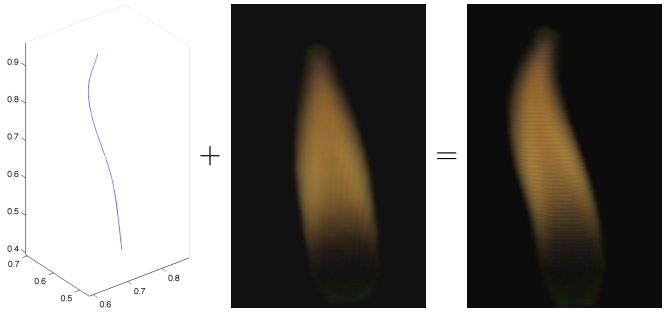


Figure 4-8: Elements of the two flames from Figure 4-7 are combined

One way of editing a flame is to take parameters from one and apply it to another. The most obvious example of this is to use the shape (core) of one flame, and the density parameters from another. Figure 4-8 combines the core from the first flame and the density from the second flame of Figure 4-7, and shows that this produces the expected result.

More direct control of the flame is also possible, which shall be explained separately for the core and the density.

4.5.1 Shape of Core

When editing a pre-existing flame, the easiest way to manipulate it is to change the shape of the core. This creates an entirely new shape of flame that still has a similar appearance of the original.

The shape of the core is defined by the function $c\colon [0,1] \to (x,y,z) \in [0,1]^3$, where $c(0)$ gives the point at the base of the core, and $c(1)$ gives the tip. An easy method to control this shape intuitively is to use an interpolating curve. Manually define a number of control points, and then a parametric curve can be used to interpolate these points. There are several options to achieve this, and they are all valid in the sense that they will produce a suitable function that defines the shape. A few options include Hermite spines, Bezier spines, Catmull-Rom splines, or B-splines [CR74, DB78]. These curves are common in tools for artists, and there exist interactive tools in three-dimensional modelling software that create them. Curves produced using these would be suitable for creating and editing flames.

An example of two licks of flames which have been shaped to look like a heart is given in

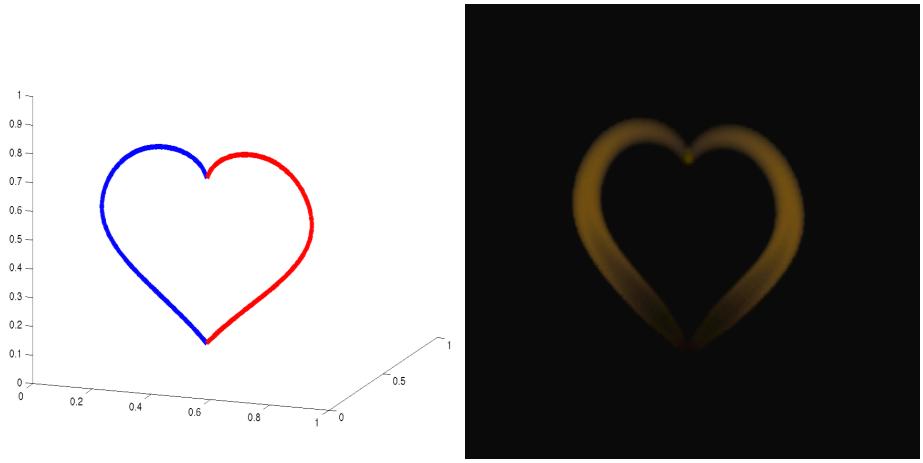


Figure 4-9: A flame (right) with its cores (left) shaped to look like a heart

Figure 4-9.

4.5.2 Density

Editing a flame's density is more complicated than simply changing the overall shape. However, the flame core model still makes this process intuitive.

Similarly to the core, the density of the flame is defined by a function $d : [0, 1] \rightarrow (\mu, \sigma, u, m, s) \in \mathbb{R}^5$, where $d(0)$ gives the parameters for the density function at the base of the core, and $d(1)$ at the tip. Again the simplest way to implement this is with a collection of parametric curves, one for each parameter, defined over a number of control points along the core. There is no need for these control points to be the same between the core and the density, or even between the density parameters, but it would simplify the process.

Considering a single point along the core, it is intuitive how to define the value for c, it is simply a point in a three-dimensional space. The density parameters are not so obvious, so their visual effect is described here.

The first parameter, μ, gives the median of the normal distribution-like shape that is used to define the density. In the context of the flame, it is distance of the peak brightness away from the core. In other words, this gives the width of the flame. A larger μ creates a wider flame, as demonstrated in Figure 4-10.

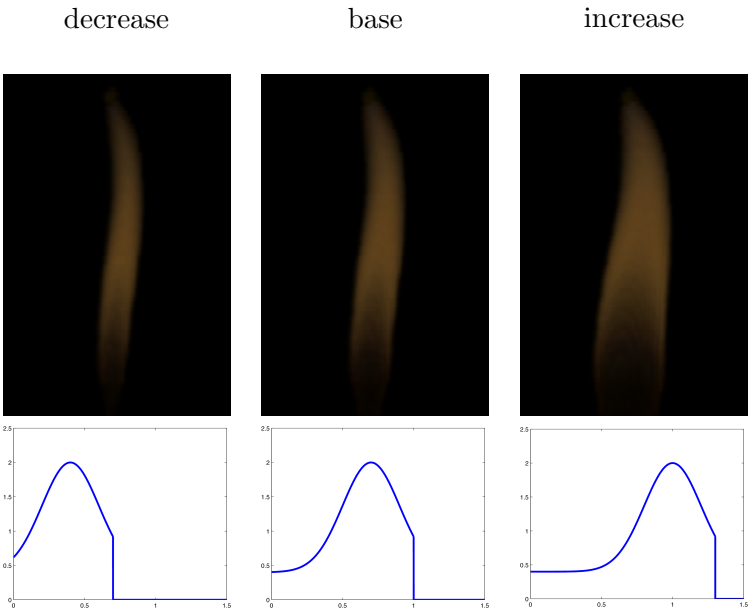


Figure 4-10: The effect of changing the μ parameter

The σ parameter controls the 'standard deviation' of the density shape. This changes the rate of decline between the peak brightness and the edge of the flame, visually allowing a more fuzzy appearance (for large values) or a sharp edge (for small ones). It also controls the incline, which manifests in the centre of the three-dimensional flame. This is less visible, but occasionally a flame appears to particularly hollow. A small value for this parameter accommodates this effect. The effect of changing σ is shown in Figure 4-11.

The m parameter is closely linked to μ and σ, as it helps define the edge of the flame. Any region beyond a distance of m from the centre will be set to zero, so setting m slightly above μ allows a very sudden drop off in density, creating a sharp edge to the flame even if σ is large. Alternatively, setting m to be far away from μ gives a softer edge, as the natural curve takes effect. While the concept is intuitive, note that this may create a non-orthogonal effect, where decreasing m may seem to undo the work of tuning σ. For this reason, the value of m should not be modified directly, and instead consider $m' = {}^{(m-\mu)}/_{\sigma}$ (this is the Mahalanobis distance [Mah36]). m' gives the distance away from the peak of the density in terms of σ. When creating or modifying a flame, set a value for m', and the value of m should be recalculated whenever σ, μ, or m' is changed. This is shown in Figure 4-12.

The u parameter gives a uniform contribution of brightness across the flame, to account

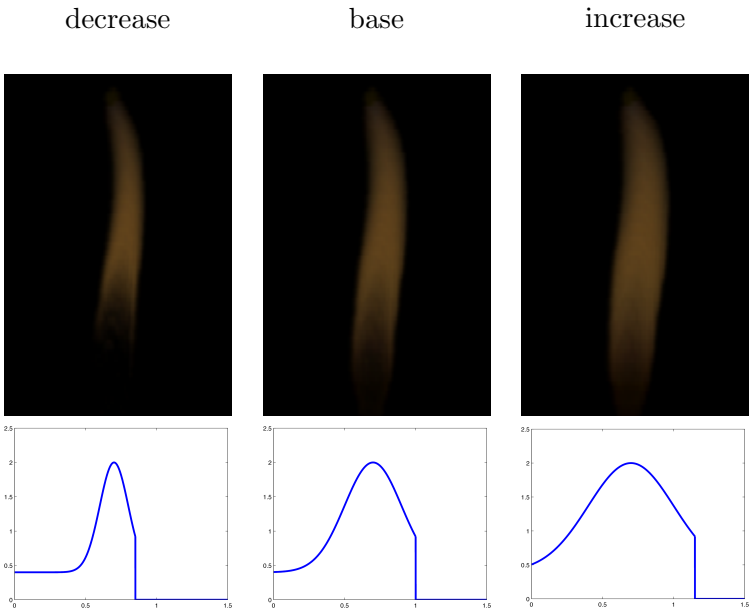


Figure 4-11: The effect of changing the σ parameter

for glowing soot particles that are dispersed throughout the volume, particularly those glowing in the centre. Changing this increases or decreases the contrast of the flame, as can be seen in Figure 4-13.

Finally the scale parameter, s, gives an overall brightness control to the flame. Changing it scales the result of the density function, creating a dimmer or brighter flame, shown in Figure 4-14.

In colour flames, both u and s are composed of three values, one for each colour channel. It is the s parameter that defines the overall colour of the flame, however it is possible to change the contrast separately for each channel via the u parameter also.

Unlike the three-dimensional line modelling tools that could be used to create a core, the density must be created manually, and it is possible to choose parameters that cause a very unflamelike appearance. It is therefore much easier to take a density from a flame that has already been created, possibly from real data (explored in the next chapter), and modify it appropriately.

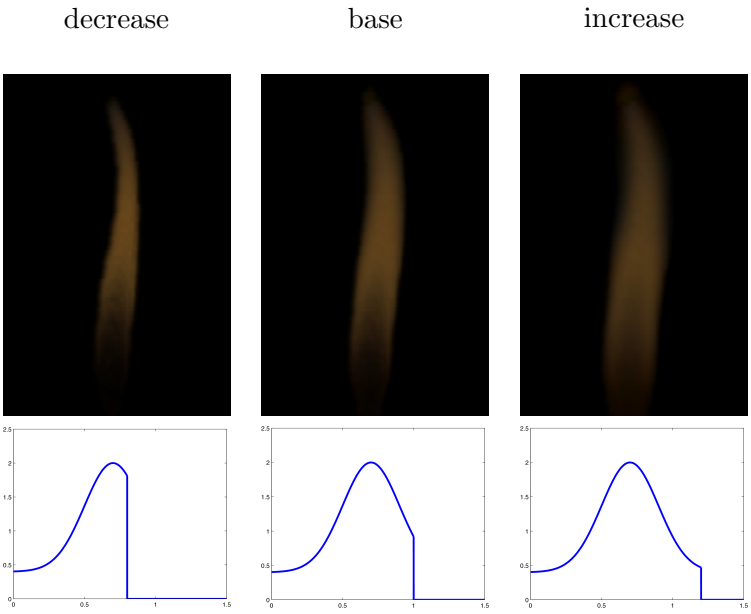


Figure 4-12: The effect of changing the m' parameter

4.6 Conclusion

In Chapter 1 the hypothesis of this document was stated as: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* This chapter presents the model that substantiates this statement and gives the foundation of all later contributions. The flame core model gives a simple structure for generating flames, the general shape is defined by a single line. The density is given by a combination of a Gaussian-like component for light at the edges, a uniform component for light in the centre, and a step function that allows a sharp drop-off in intensity. While this density function was designed based on brightness, simple comparisons to prior work on thermal imaging of flames suggest that it gives a good basis to model heat as well [Sch96]. This is no surprise; in a yellow flame brightness and heat will be closely linked, since the majority of light is coming from hot glowing soot particles.

The flame core model, although conceived independently, is similar in concept to a prior work: Structural Modelling of Flames (Section 2.2.5). As it was developed by a film studio, it gives results that are of a high enough quality to be used in high budget animated films. The work in this chapter differs by using a set of simple parameters to explicitly define the appearance, rather than relying on random particles and noise functions. This means

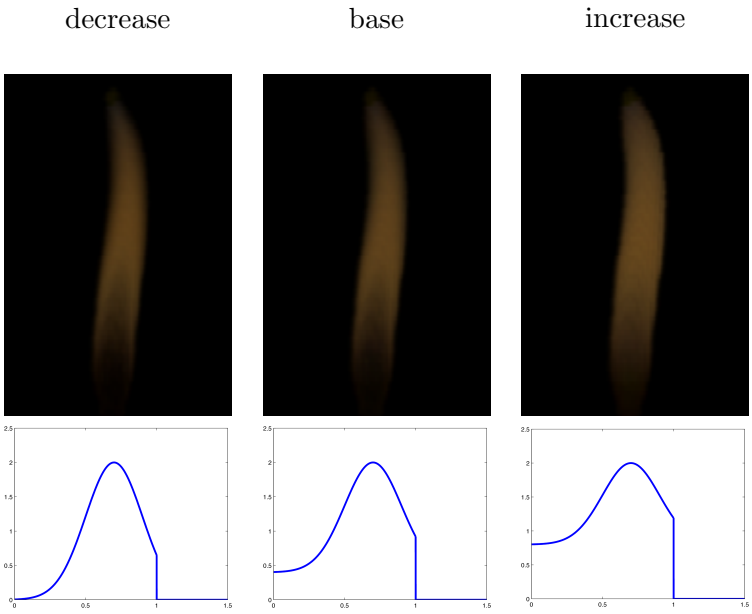


Figure 4-13: The effect of changing the u parameter

examples in the flame core model offer direct intuitive control over their appearance, as demonstrated.

It is because of this explicit representation that the model is well suited to being fit to real video data, as Chapter 5 will show next, completing the goal of the original hypothesis.

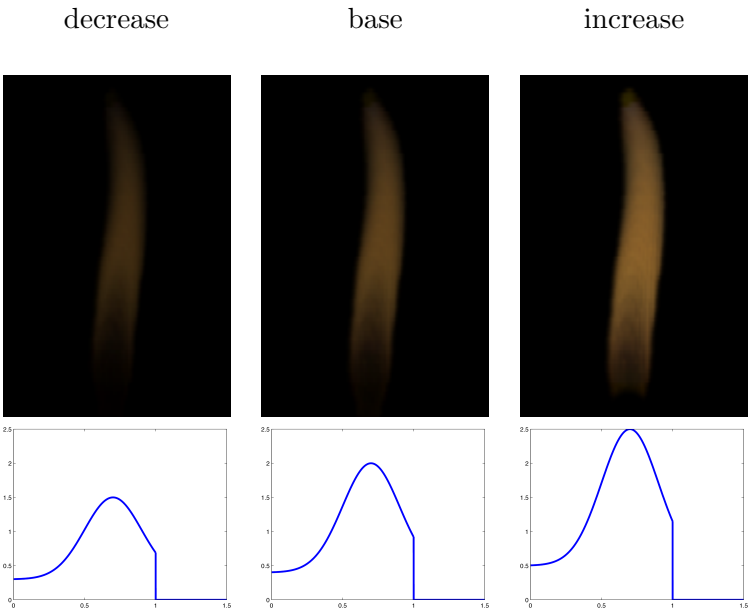


Figure 4-14: The effect of changing the s parameter

Chapter 5
Fitting Cores to Acquired Data

This chapter details how the flame core model (Chapter 4) can be controlled by real flames – specifically the three-dimensional models detailed in Chapter 3. It also presents an early approach for fitting this model to a single uncalibrated image of a flame, which is novel for the field.

In cases where the input is not ideal for the model, a partially-interactive approach is given to pre-process the volumes. It is shown that the results visually match the input and look like realistic flames.

5.1 Overview

The goal of the fitting process is to find the flame core parameters which minimise the error between the reconstructed flame and the original volume. That is,

$$\underset{(c,d)}{\operatorname{argmin}} \sum_{x,y,z=1}^{N} \left(\varphi(x,y,z) - \phi(\delta,\mu,\sigma,u,m,s)\right)^2 \tag{5.1.1}$$

where φ gives the true density at the point (x, y, z) from the input model. ϕ gives the reconstruction density at this point, which is defined along with the other variables in Section 4.4: c gives the core of the flame, δ represents the distance from the point (x, y, z) to the closest point on the core c, and d gives the density parameters μ, σ, u, m, and s.

This problem is difficult to solve; the search space is large, and the parameters do not behave linearly: if the core is changed, then this may result in a different closest point,

which would mean a different set of density parameters. For this reason, the core of the flame is determined first, and then the density parameters are found second.

There are several other challenges when fitting to real data. The method must be robust, accepting flames of different types, even those that do not fit the ideal conditions for the model. Since the input is assumed to be a real flame that is from a video source, the shape will be partly determined by the shape of the burning material. The shape of a flame produced by a candle will be markedly different from one produced by a dish of alcohol. The flame core model is most applicable for modelling tall, thin licks of flame. If the source of the flame has a wide base, then the flame will have an unmoving, short, wide base as well, and this will be unsuitable for modelling with the model. Furthermore in these situations, licks of flames will move around erratically and overlap; the gases will combine and create the appearance of a single solid flame, sometimes with odd properties that do not fit the usual appearance solely from convection. These licks can still be modelled as separate cores, however this will require identifying them as separate licks from the original data, and extracting the information relevant to each one.

Once the individual licks have been extracted from the flames, a core is fitted to each one. The method must be able to create cores that are smooth and match up with what is intuitively recognised as the 'centre' of the lick. This must work with licks that bend in unusual ways, sometimes appearing to completely change direction. They should not break or distort at points where two licks of flame meet, and should be resilient to noise or missing information in the original data.

Then, the density parameters for each lick should be found. This requires identifying the density in terms of distance from the core, then finding a set of parameters for the density function (Equation 4.3.3 from the previous chapter) that match this. While this is a more straightforward curve fitting problem in theory, there are some practical complications which are detailed as well.

Finally a method for fitting the model to a single uncalibrated two-dimensional input is also given, in this case the objective is the two-dimensional equivalent of Equation 5.1.1, with an added projection which is discussed later.

5.2 Algorithm

It is assumed that the input data is given as voxel models, which can be thought of as three-dimensional images. This is equivalent to the format used in [IM04], which is given as a set of weights a_i over a collection of basis functions ϕ_i^{box} (see Section 2.4.2). Colour flames are given as three such voxel models, one for each colour channel (red, green, and blue).

There are two sources of high quality voxel model flames from video at the time of writing. The first is the work that provided the reconstruction method [IM04], and the video is of a burning dish of alcohol. The second is the data that was captured as part of this project. This contains videos of a candle and a lighter. As mentioned, tall single-lick flames are more easily modelled. Hence the algorithm changes slightly depending on whether the flame is simple (candle, lighter) or complicated (dish of alcohol). The main difference is the pre-processing required to extract the licks from the larger flame. More detail about these input flames is given in Chapter 3.

5.2.1 Pre-process Data

The first step of the process is to convert the voxel format into a collection of points, or point cloud, for further processing. A discrete number of points per voxel location are chosen based on the density in each voxel, high densities result in more points. This is likely to introduce some loss of information depending on the storage format of the original densities, however it can be controlled quite easily. The original densities are scaled by some factor s and rounded to the nearest integer, then the scaled density at each voxel gives the number of points at this location. In testing it was found that $s = 100$ captured the detail of the flame sufficiently, i.e. the maximum number of points is 100, for a voxel with density 1.

The possibility of jittering the points within each voxel by some random amount was tested, however it did not improve the results of any subsequent density related analysis, but it did worsen repeatability and running times.

Now a flame is given by a multiset of points $\mathbf{F} = \{\mathbf{p}_1, \mathbf{p}_2, \dots\}$ where $\mathbf{p}_i = (p_x, p_y, p_z) \in \mathbb{R}^3$. Colour flames will give three such sets: $\mathbf{F}_{\text{R}}, \mathbf{F}_{\text{G}}, \mathbf{F}_{\text{B}}$, and a fourth set $\mathbf{F}_{\text{K}}$ corresponding to the points given by a greyscale version of the flame should be found as well. If they are not

already, the points are scaled by the voxel resolution so that they lie in $[0, 1]$. The points are not tightly bound into the unit cube, so aspect ratio and translations between frames are preserved. Some noise reduction proved useful at this point also: simply removing any points which did not have any neighbours from adjacent voxels was sufficient.

5.2.2 Identify Licks

The goal of this part of the algorithm is to split the flame into its licks. For very simple flames, this step is unnecessary, since the entire flame can be treated as one lick. The following is the process for identifying licks in a larger or more complicated flame.

For colour flames, this step is performed on the greyscale set of points $\mathbf{F}_{\mathrm{K}}$. Once these points have been divided, it is a simple procedure to find corresponding points in the colour point sets. Any points from $\mathbf{F}_{\mathrm{R}}, \mathbf{F}_{\mathrm{G}}$, and $\mathbf{F}_{\mathrm{B}}$ that are not in $\mathbf{F}_{\mathrm{K}}$ are either given the label of the closest point, or simply discarded.

If the flame has a constant base, such as a flame produced by a dish of alcohol, then this should be identified separately from the licks, and then later removed. An initial guess is found by culling any points that are below a user-defined constant height. i.e.

$$\mathbf{F}' = \{\mathbf{p}_1, \mathbf{p}_2, \cdots \in \mathbf{F} \text{ such that } p_z > h\}. \tag{5.2.1}$$

The constant h is chosen to be big enough to cut out the entire base.

Then, the points are separated into connected components. Since the points originated from voxel data, this is easy to do. Simply pick a random point, and label all points within distance d as neighbours. Now do this repeatedly, labelling all neighbours of neighbours, until there are no more points within this distance. If there are still points unlabelled, then repeat the entire process (starting with a new label) until every point is labelled. Assuming N is the number of voxels per dimension, then for 6-connected points $d = 1/N$, for 18-connected points $d = \sqrt{2}/N$, and for 26-connected points $d = \sqrt{3}/N$.

Now each point has a label, which identifies it as belonging to a particular lick. The points that were removed as part of removing the base are given a new unique label as well. This results in a partitioned flame as required, with each lick and the base being identified by separate labels. However the cut between the licks and the base will be a perfectly horizontal plane, which will create an unrealistic appearance to the lick, and may miss points that lay below the cut-off. So, these labels are used as an initial state for the

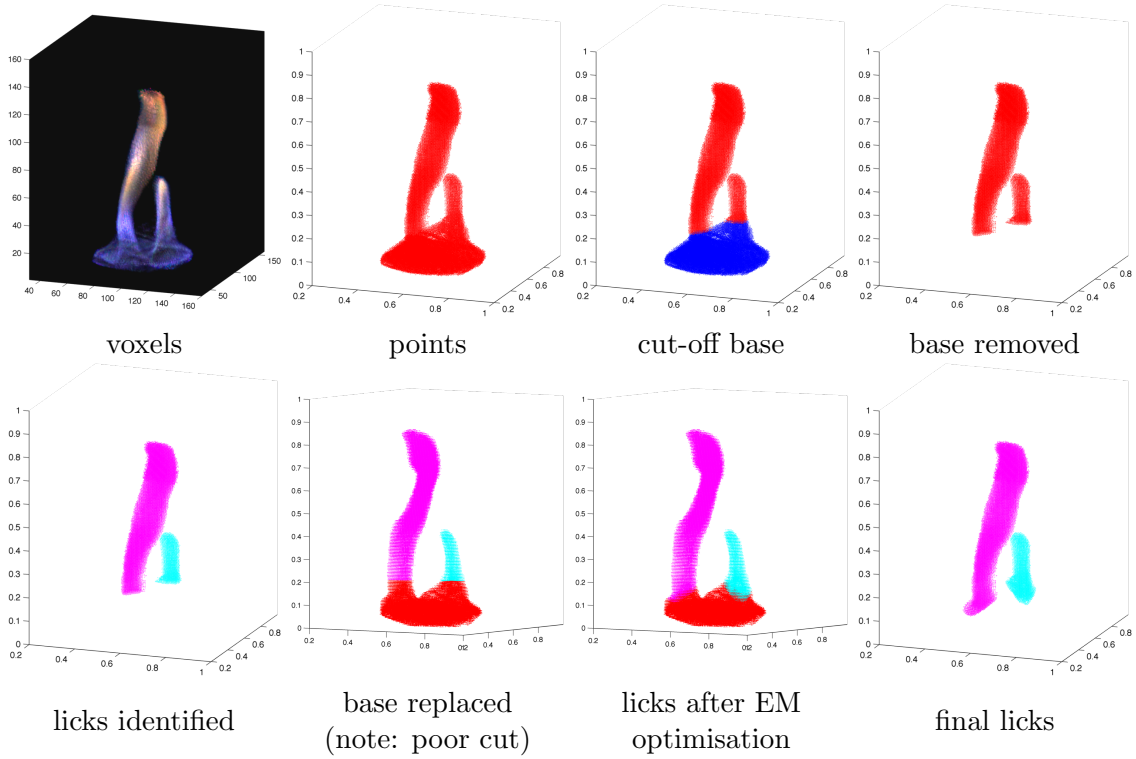


Figure 5-1: A flame is split into its component licks and the base is discarded

EM algorithm [DLR77] to build a Gaussian mixture model. This finds a more natural cut between the base and the licks, and gives the final state of the partitioning.

Due to the tall thin nature of flames, it was found that compressing the flames in the upwards (z) axis before applying the EM algorithm resulted in a much better fit. i.e. use $p_z^* = \frac{p_z}{s}$ where s is a scale value. s can be found as roughly how much taller the flames are than they are wide, normalising the aspect ratio before performing the clustering.

Separating Merged Licks

In some cases, licks of flame are intersecting and so are not identified as being separate by the simple connected component algorithm in the previous section. In these cases, a user provided value can be used as the target number of licks (which is evaluated from the source data). Then, the base of the flame is removed as usual (if necessary), and then mean shift segmentation [CM02] is used to get the desired result.

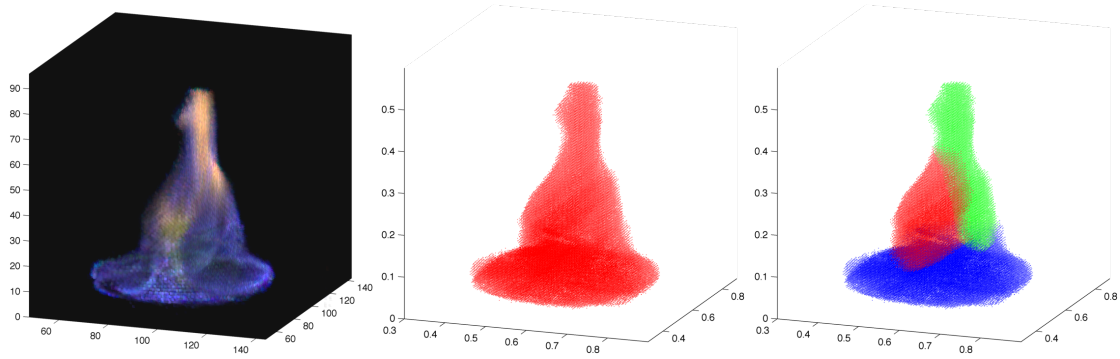


Figure 5-2: Mean shift finds a natural split between licks that appear joined

5.2.3 Core

Once the flames have been split into licks, each one is considered individually, and the parameters of the model are found. This starts with the structural element of the lick, the core. The goal is to find a smooth line rising upwards through the centre of the lick of flame, as illustrated in Figure 4-6. A number of existing methods were applied to the data, however none gave a satisfactory result, and so a novel method was developed.

The first ideas tested were a number of skeleton methods. The concept of a skeleton was originally defined by Blum in 1967 [Blu67] as a thin figure centred in a shape, which summarises its form, and preserves the topology and connectedness of the shape. Methods typically operate over binary images or volumes, rather than considering the density of the shape. The main reasons these methods failed were due to sensitivity to changes in the original volume, and the tendency to create a branching structure. Hence, the results were too complicated to use as a core, or they did not produce a skeleton-like result at all from the licks of flame. This was the case for principle curves [HS89a], and the iterative thinning skeleton algorithm [HS91] applied in three dimensions. Approaches from medical imaging on centreline extraction from three-dimensional volumes are also similar [LKC94, GS97, BST05]; they assume that structures have little to no change in width along the centreline, which is clearly not true of licks of flame. This was also true of another method that generated skeletons directly from point clouds [TZCO09].

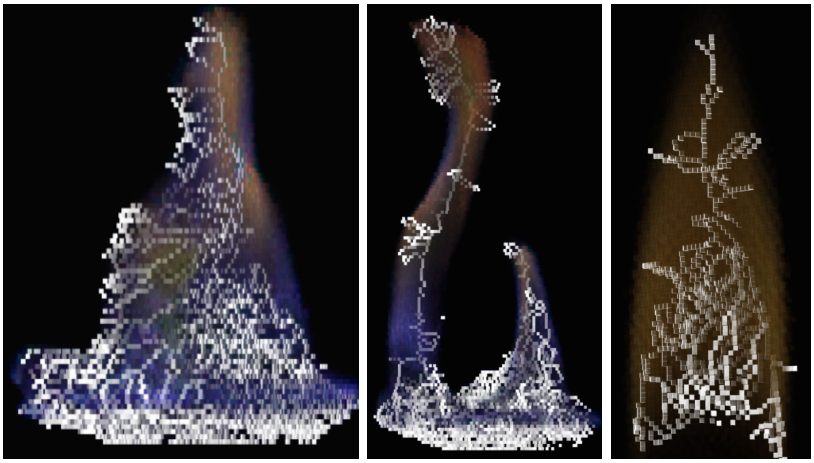


Figure 5-3: A standard skeleton method generally overfits, but performs best in tall thin areas, as the central image shows

Core Fitting

Instead, a novel method was developed to find the desired result. The basic idea is to take horizontal slices of points through the lick, and find the mid-point of each one. This is done as follows. First, quantise the points in the z (vertical) direction: assuming the lick of flame is given as a set of points

$$\mathbf{L} = \{\mathbf{p}_1, \mathbf{p}_2, \dots\},\ \mathbf{p}_i = (p_x, p_y, p_z) \in [0,1]^3 \tag{5.2.2}$$

then take

$$p'_z = \left\lfloor \frac{p_z}{N} \right\rfloor N \tag{5.2.3}$$

where $\lfloor x \rfloor = \max(\{m \in \mathbb{Z} \text{ such that } m \leqslant x\})$ is the typical floor function, but could equivalently be round or ceiling functions, and N is a constant that controls how granular the quantisation should be. N should be at most the number of voxels per dimension in the reconstruction, but can be made lower to increase speed or smoothing. $n = 1/N$ is the distance between quantised points in the z direction.

Then the points on the core are given by the mean point in the horizontal plane (mean x and mean y) for each quantised z

$$\mathbf{C} = \{(\overline{\mathbf{s}_x}, \overline{\mathbf{s}_y}, z)^\mathsf{T}\}, \quad \text{for all } z = z_{\min}, \dots, z_{\max} \tag{5.2.4}$$

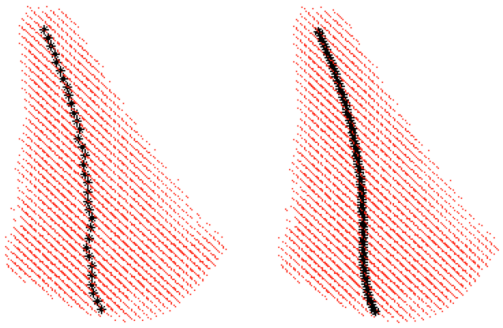


Figure 5-4: Interpolating the points is necessary to give a smooth result

where $\mathbf{s}_{[x,y]}$ gives the x or y points that correspond to each horizontal slice,

$$\mathbf{s}_{[x,y]} = (p_{[x,y]} \text{ such that } p'_z = z \text{ and } \mathbf{p} \in \mathbf{L}), \tag{5.2.5}$$

and $\overline{\mathbf{x}}$ is the arithmetic mean of an array of points

$$\overline{\mathbf{x}} = \frac{1}{m} \sum_{j=1}^{m} x_j. \tag{5.2.6}$$

This produces a set of points that runs up the centre of the lick. Naturally the points will not lie on a smooth curve – as with the skeleton methods, this is currently still sensitive to boundary changes or noise. So, these points are interpolated, specifically with a piecewise cubic spline, creating a smooth result which still approximates the centre of the lick (see Figure 5-4).

This technique works well for licks of flame that have a vertical orientation. However, if a lick is bending or swaying in one direction, such as a result from wind, this approach will produce results that are offset from the visibly true centre of the flame. A simple solution is to rotate the points before the initial quantisation, so that the most-upwards principal vector [Per01, Hot33] $\mathbf{v}$ is pointing directly upwards. In other words, find the rotation matrix R that solves

$$(0, 0, 1)^{\mathsf{T}} = R\mathbf{v} \tag{5.2.7}$$

This matrix is found using Möller and Hughes' method [MH99]. The rotation is in the plane containing both vectors. Then, use this matrix to rotate each point in the lick $\mathbf{L}$, use the method above to find the points along the core for the rotated points, and then

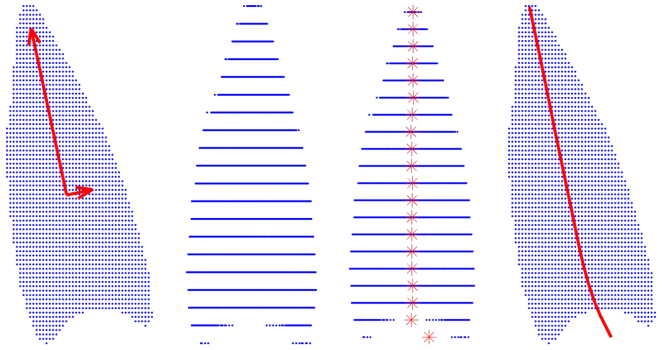


Figure 5-5: Left to right: find upmost principle direction, orient lick and quantise, find mean point of each slice, then smooth and rotate back

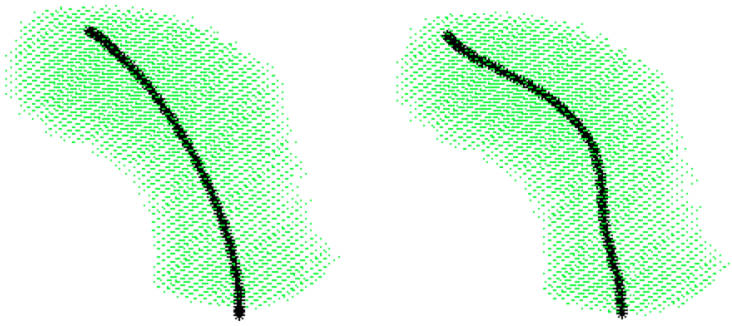


Figure 5-6: Right image shows the result of rotating the lick before finding the core, left shows the the result without any rotation

rotate these points back using the inverse of R. An illustration of the method is shown in Figure 5-5, and a comparative result can be seen in Figure 5-6.

Local Orientation Method

The method that obeys the global orientation provides a good solution for most cases, especially on simple licks. However, some particularly curved flames still produce less than ideal results, because the overall orientation of the lick is not specific enough to refine the result. What is required is a method that matches the core to the curvature of the points locally. Thus, an iterative method was designed which 'grows' a core from the centre of the lick out towards the ends, following the curvature. To reduce the chance of overfitting, and to save computation time, this method is only used when the previous one is found to be a poor fit.

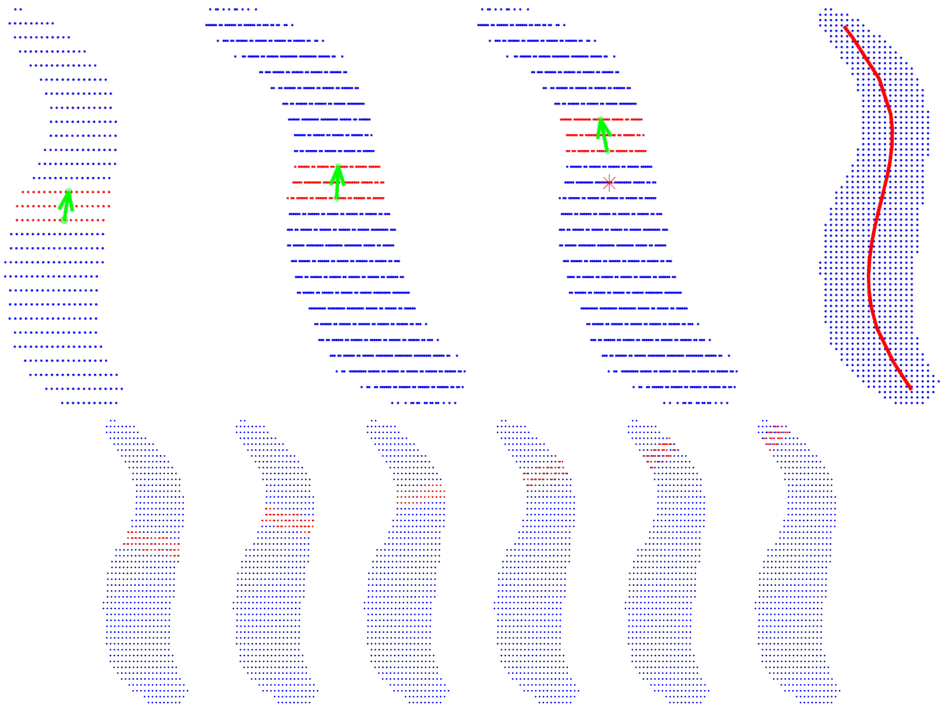


Figure 5-7: Top: find a horizontal disc of slices, orient the lick according to the direction (green arrow), continue with the next slice, the final result
Bottom: from the original orientation, the discs bend around the curve

The basic idea is to take a series of discs, each one consisting of several horizontal slices of quantised points. These are rotated to suit their orientation, and a core point is found for each disc. Take a disc: the centre points of the top and bottom slices form a vector which would point directly upwards if they were perfectly aligned. Most of the time this will not be the case, and so the points are rotated to align this vector with the upwards direction. The points are requantised, and since the rotation may have introduced new points to the disc, the slices are recalculated. The process repeats until the slices are aligned, or the rotation makes it worse. A point is taken at the centre of the disc as a point on the core, the disc is moved up the lick, the disc is aligned, and so on until the entire lick has been processed. By considering discs in the lick sequentially, the process can 'turn' corners, giving a better fit when the flame is particularly curved. The method is illustrated in two-dimensions in Figure 5-7.

First, as before rotate the points using the matrix R from Equation 5.2.7, call this R_0,

and call the set of rotated points $\mathbf{L}_0$, i.e.

$$\mathbf{L}_0 = \{R_0\mathbf{p} \text{ for all } \mathbf{p} \in \mathbf{L}\}. \tag{5.2.8}$$

This gives the initial estimate for the orientation of the lick. The quantised z points are found as before, using Equation 5.2.3.

Next, taking a value that is half way up the lick $z_{\text{mid}} = \frac{z_{\min}+z_{\max}}{2}$, find the disc of points

$$\mathbf{D}_0 = \{\mathbf{p} \in \mathbf{L}_0 \text{ such that } z_{\text{mid}} - in \leqslant p_z \leqslant z_{\text{mid}} + in\} \tag{5.2.9}$$

where i is a constant that controls how many slices of the lick to use in each disc. A lower number is more granular, but risks overfitting. $i = 1$, giving 3 slices per disc, was found to work well in these curved flames.

This disc will give a single point on the core, but first it is oriented so that the vector between the top and bottom slice is aligned with the z axis. The disc points are recalculated, and this process repeats until convergence. This requires a few lines to detail mathematically, but the concept is simple. To find the top and bottom slice in each disc, take the mean points in the x and y planes, giving the centre points of these slices

$$\mathbf{d}_{\max} = (\overline{\mathbf{s}_x}, \overline{\mathbf{s}_y}, z_{\text{mid}} - in)^{\mathsf{T}} \tag{5.2.10}$$

$$\mathbf{d}_{\min} = (\overline{\mathbf{s}_x}, \overline{\mathbf{s}_y}, z_{\text{mid}} + in)^{\mathsf{T}} \tag{5.2.11}$$

where again, $\mathbf{s}$ represents the x or y values in the slice corresponding to the given z value, and $\overline{\mathbf{s}}$ is the arithmetic mean.

Now, take the vector between the top and bottom midpoints

$$\mathbf{v}_1 = \mathbf{d}_{\max} - \mathbf{d}_{\min} \tag{5.2.12}$$

and calculate the rotation matrix that aligns this vector with the upwards primary axis, as in Equation 5.2.7:

$$(0, 0, 1)^{\mathsf{T}} = R_1\hat{\mathbf{v}} \tag{5.2.13}$$

where $\hat{\mathbf{v}}$ is the normalised vector:

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} \tag{5.2.14}$$

This new R_1 is used to reorient the points. Apply this rotation matrix to get a new $\mathbf{L}_1$, and then the whole procedure – quantise, find $\mathbf{D}_2$, find $\hat{\mathbf{v}}$, find R_2 – is repeated until convergence ($R_t = I$, where I is the identity matrix) or the angle between $\hat{\mathbf{v}}$ and $(0, 0, 1)^\mathsf{T}$ is greater than the previous iteration, in which case roll-back one iteration and take the previous R_t as the final orientation for this disc.

It is important to calculate the rotated points from the original points $\mathbf{L}$ at each iteration. If the quantised points are used, this will obviously lead to drift due to repeated rounding operations. It is wise to use the originals even if the non-quantised points could be used, due to the possibility of accumulating floating point errors. On the first iteration calculate $R^* = R_1 R_0$ which gives the rotation matrix from the original points $\mathbf{L}$. At each step a new rotation can be found, $R^{**} = R_2 R^*$, and so on.

Once the above sequence terminates, the centre point of the final disc $\mathbf{D}_t$ is recorded as a point on the core. This is

$$\mathbf{c}_i = \mathbf{d}_{\text{mid}} = (\overline{\mathbf{s}_x}, \overline{\mathbf{s}_y}, z_{\text{mid}})^\mathsf{T} \tag{5.2.15}$$

Next, z_{mid} is moved upwards by a fixed amount so a disc can be found higher up the lick

$$z_{\text{mid}} = z_{\text{mid}} + kn \tag{5.2.16}$$

(where typically $k > i$, but is not necessary). Now the whole process is repeated: find a disc of slices, orient the disc, and add a point to the the core. The last rotation matrix is used as the starting rotation for the next z_{mid}.

After finding each core point $\mathbf{c}_i$ and before moving up the lick, calculate the distance to the edge of the lick at this slice, and the distance to the topmost point of the lick

$$\delta_{x-y} = \max_{\mathbf{p} \in \mathbf{D}^i} \left\| \mathbf{p}_{\{x,y\}} - \mathbf{c}_{\{x,y\}} \right\| \text{ and} \tag{5.2.17}$$

$$\delta_z = \max_{\mathbf{p} \in \mathbf{F}'} |p_z - c_z| \tag{5.2.18}$$

If $\delta_z \leqslant \delta_{x-y}$ then this is the tip of the core. The topmost point of the lick is closer than the width, and the process should stop moving upwards.

Now return to the midpoint of the lick, the original z_{mid}, with the original rotation R, and move downwards to the base of the lick. This time, continue until $\mathbf{D}_t = \emptyset$.

The reason for the disparity in termination conditions for the up and down directions is

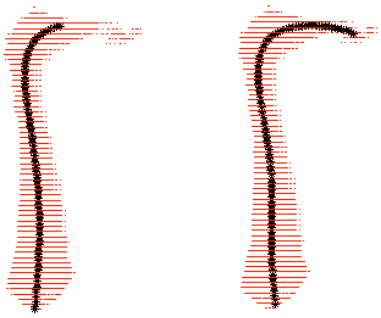


Figure 5-8: Left shows method obeying global orientation, right shows method obeying local orientation

due to the way the rendering algorithm achieves the visual look of the top and the bottom of the lick, in particular flames which have a more rounded top rather than coming to a point. See Section 4.4.1 for more details.

Now, there is a collection of core points $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots\}$, and the final core $c(u)$ is given by fitting a parametric curve to these points, as previously.

The time taken to compute the solution of either method depends on the number of points in the lick, which is determined by the original voxel resolution and the value for s chosen in the pre-processing (Section 5.2.1) which gives the number of points for a given density. For the candle dataset, the resolution gave about 900,000 points in each lick. Using this, the global orientation method takes about 10 seconds to fit a core, and the local method takes about 25 seconds. In both cases, the runtime appeared to scale linearly. This is consistent with the fact that it is only a one-dimensional problem.

5.3 Density Parameters

Once each lick of flame has its core, the remaining parameters correspond to the density of the flame around this core. Recall the density is defined by the function $d\colon [0,1] \to (\mu, \sigma, u, m, s) \in \mathbb{R}^5$, and that for colour flames the u and s parameters have three components, one for red, green, and blue. Whether the flame should be colour or not, the first step is to find these parameters assuming it is going to be greyscale. Then the colour parameters are found second if necessary.

The density function that uses these parameters (4.3.3) is as follows:

$$\phi(x,\mu,\sigma,u,m,s) = \begin{cases} s\left((u-1)\exp\left(-\left(\frac{x-\mu}{2\sigma}\right)^2\right)+u\right) & \text{if } x \leqslant m \\ 0 & \text{otherwise} \end{cases} \tag{5.3.1}$$

Therefore, with the real flame data in the right format, finding suitable density parameters becomes a curve fitting problem. To solve this, the Levenberg-Marquardt algorithm [Lev44, Mar63] is used. This is a well established method for least-squares curve fitting, which combines gradient descent with a dampening term. The fitting problem, then, is to process the real flame data into the correct format, and then find suitable starting parameters.

First, for each point in this lick $\mathbf{p}_i \in \mathbf{L}$, find the closest point on the core

$$h_i = \underset{x\in[0,1]}{\operatorname{argmin}} \|c(x) - \mathbf{p}_i\| \tag{5.3.2}$$

and denote the corresponding distance

$$\delta_i = \|c(h_i) - \mathbf{p}_i\|. \tag{5.3.3}$$

In order to represent the changing density along the core from the finite number of points that represent it, the values h_i are put into bins, and a set of parameters are found for each bin. This is analogous to the process taken when creating a histogram. The number of bins gives how smooth or granular the density parameters will be, and can be determined from the arc length of the core: let

$$l = \int_0^1 \sqrt{1 + c'(u)^2}\,\mathrm{d}u \tag{5.3.4}$$

be the length of the core, then the width of each bin should be the length divided by the size of a number of voxels – for instance, to average over three voxel widths, choose the bin size $b = l/3n$ where n is the width of a single voxel. Alternatively the number of bins can be set to a constant that gives a pleasing smoothness.

For some bin width b, the bins are given by

$$\mathbf{B}_k = \{\mathbf{p}_i \in \mathbf{L} \text{ such that } kb \leqslant h_i < kb + b\},\ k = 0, 1, 2, \ldots \tag{5.3.5}$$

Density parameters are found for each bin. The distances from the core are given by the δ_i corresponding to each point $\mathbf{p}_i$, and these give the x-values for the curve fitting. The corresponding densities, the y-values, can be found in two ways

- return to the original voxel model and find the density values for each corresponding point in this bin; or
- for each point in this bin, count the number of identical points, and divide by the scale factor used when converting from the voxel representation to points.

This gives a collection of points in the form (δ, d), distance from core vs. density at this distance. Since the flame is circular, there will be multiple values of d with the same δ. Furthermore, since the core will likely not lie directly in the centre of a voxel, the distances will irregularly spaced. Before running the algorithm, this data is also split into bins in a histogram-like manner. This time the distances from the core are split, with the bin size based on the size of a voxel, $b = 4n$ was found to provide good results.

$$\mathbf{Y}_k = \{d_i \text{ such that } kb \leqslant \delta_i < kb + b\},\ k = 0, 1, 2, \ldots \tag{5.3.6}$$

This gives the data used for the curve fitting, each bin gives: the mid-point of the distance corresponding to this bin vs. the mean of the density points contained in this bin.

$$(kb + {}^{b}/_{2}, \overline{\mathbf{Y}_k}),\ k = 0, 1, 2, \ldots \tag{5.3.7}$$

Here the arithmetic mean of a multiset $\overline{\mathbf{S}}$ is defined as one would expect, analogously to that of an array (Equation 5.2.6).

Clearly the maximum value for the distance δ is associated with a non-zero density d – there will be no bins for distances further away than the data that exists. It is helpful for the curve fitting of the step function to add extra points for equally spaced, imaginary bins above this distance with zero density.

The difference between the data before and after this averaging and adding of zeros is shown in Figure 5-9.

Next, initial estimates for the density parameters are required. These are as follows:

- m is the distance of the last point before any were added, $m = \max(\delta_x)$;
- s is given by the peak density, $s = \max(d_x)$;

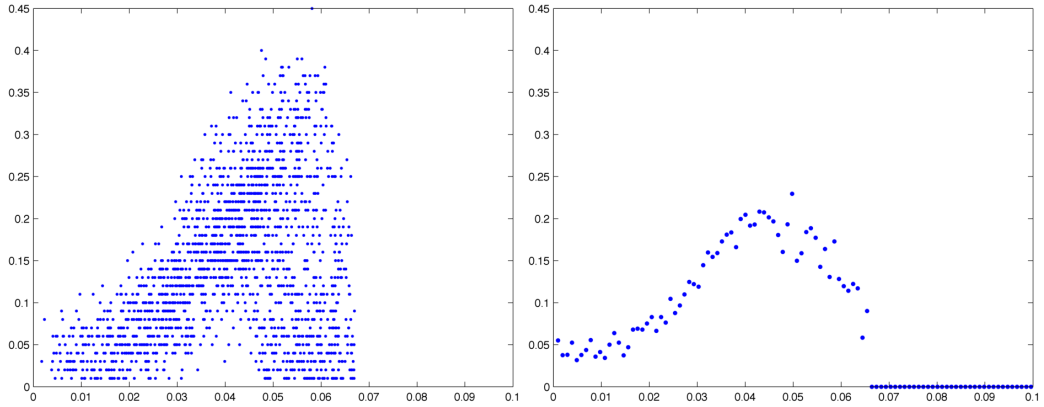


Figure 5-9: An example of curve fitting data, before and after processing

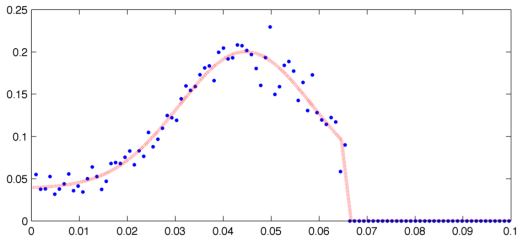


Figure 5-10: Data from Figure 5-9 with curve fitting result shown in red

- μ is given by the distance of the peak density, $\mu = \delta_x$ where $x = \text{argmax}_x(d_x)$;
- u is the ratio of the maximum and minimum values of the density, $u = {}^{s}/_{\min(d_x)}$;
- σ is just a small estimate, for example $\sigma = 0.1$.

Then the Levenberg-Marquardt algorithm is applied until the parameters converge. Once this entire process has been completed for each bin of points along the core, the fitting is complete. The values of $d(h)$ (the density parameters along the core, from the definition) are given by the parameters that were found in the curve fitting step, and a parametric curve can be fit to the results, as in the construction of the core itself. An example of a curve that was fit to a real flame density is given in Figure 5-10.

If the desired output is in colour, then the previous section is run again for each channel of the original RGB voxel model, with all the parameters except for u and s fixed. The previously-found values for u and s are used as the initial parameter estimates for their respective colour parameters.

Some results for flame core model versions of flames from Chapter 3 are shown in Figure 5-

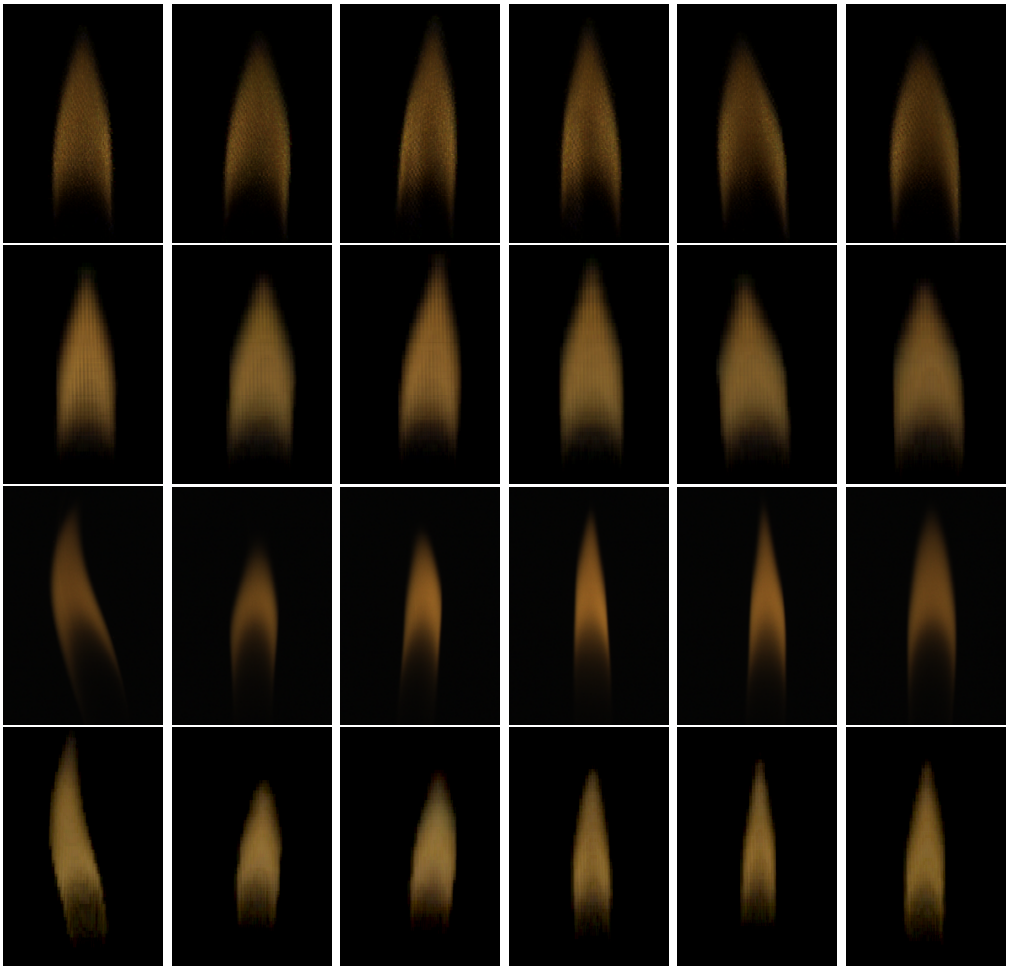


Figure 5-11: Alternating rows show original volumes and flame core renderings
Top two rows are the candle, then the lighter

11, along with the original three-dimensional models to which they correspond. These flames are more convincing when animated, and videos are contained on the enclosed disc.

The parameter fitting process has two costly steps in terms of running time. The first is finding the nearest point on the core for every point in the lick, the second and more significant is the Levenberg-Marquardt algorithm. The former depends on the number of points in the lick, but still takes less than 0.5 seconds on higher resolution flames (around 900,000 points). The latter does not, since the process runs on averaged data, and takes around 7 seconds. For colour flames, this process has to be repeated for each colour channel, but the Levenberg-Marquardt algorithm only has to find two parameters for each. The total time for a colour flame was around 15 seconds.

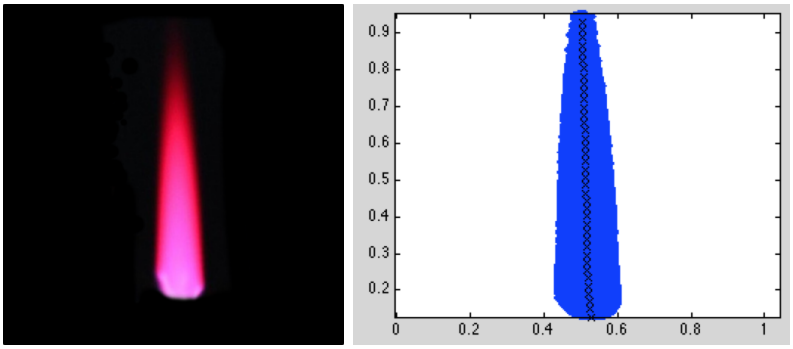


Figure 5-12: A two-dimensional image of a pink lighter flame (left), with a core (right)

5.4 Two-dimensional Input

While the flame core model was designed to be three-dimensional, it would be of great interest and use to fit flames to two-dimensional images or videos. This is particularly true if the resulting flame is in three dimensions. In previous literature on flame reconstruction, the idea of using one camera for three-dimensional capture has been dismissed entirely. In [HK03] specifically, it is stated:

> "Fire is an inherently dynamic phenomenon and, as such, requires simultaneous image capture."

However, using the flame core model, it is possible to at least approximate the appearance and shape of a flame from a two-dimensional representation. This work has not been fully explored, but a proof of concept is presented here. It is assumed that the two-dimensional image is of a simple flame (a single core), and it is viewed directly from the side (perpendicular to the direction of gravity). It is also assumed the camera calibration is unknown; using this could result in a more accurate result, which could be explored in future work.

First, as with three-dimensional input, a core is found through the centre of the flame. The techniques from the three-dimensional case reduce down to two dimensions quite simply. In Figure 5-12, a core is fitted to an image of a pink lighter flame.

Fitting the density parameters in the two-dimensional case is more complicated. Since the image is a projection, the observed density of the flame will not be the same as the three-dimensional case. This is most obviously seen by considering the centre of the flame: the assumptions previously were that the density is low, formed only by glowing

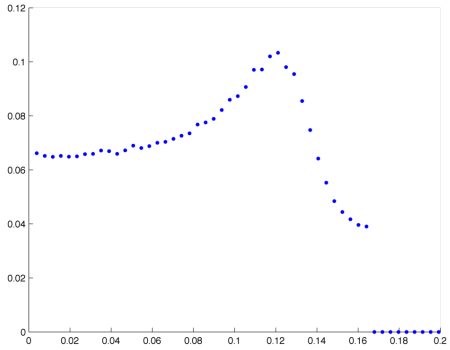


Figure 5-13: The density of a two-dimensional flame, in terms of distance from core

soot particles. However, when viewed from the side, the centre of the volume is actually the value produced by integrating over the entire density in the direction perpendicular to the plane of projection. A collection of points (d, δ) was found in the same way as the three-dimensional process, showing distance from core against density. This is illustrated in Figure 5-13. Clearly the regular density function will not fit to these points.

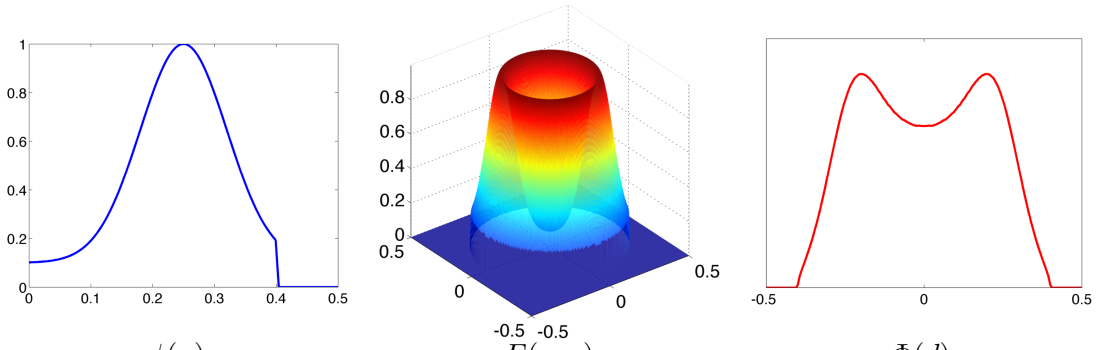
Mathematically the projection of the density function to two dimensions can be modelled as a surface of revolution, which is then integrated along the newly created axis. This model will assume parallel projection rather than perspective, which in reality will only be true if the flame is far away from the camera. This is ignored for the sake of simplifying the model.

A reminder of the density equation from 4.3.3, for a distance d from the core and given parameters μ, σ, u, m, and s:

$$\phi(d) = \begin{cases} s\left((u-1)\exp\left(-\left(\frac{d-\mu}{2\sigma}\right)^2\right)+u\right) & \text{if } d \leqslant m \\ 0 & \text{otherwise} \end{cases} \tag{5.4.1}$$

First, consider rotating this function around a vertical axis. This would create a surface, which could be expressed the form $f(r, \theta)$. Clearly the value of θ has no impact on the result, because it is radially symmetric. In fact $f(r, \theta) = \phi(r)$. Converting this to a Cartesian representation, as illustrated in Figure 5-14:

$$F(x, y) = f\left(\sqrt{x^2+y^2}, \text{atan2}(y, x)\right) = \phi\left(\sqrt{x^2+y^2}\right) \tag{5.4.2}$$



$\phi(x)$ $F(x,y)$ $\Phi(d)$

Figure 5-14: The density function (left) is rotated to create a surface (middle)
Right: numerical solution to a parallel projection through the volume
Compare numerical approximation ($x > 0$) to observed values (Figure 5-13)

So the expected form of the density function in a parallel projected image is a line integral:

$$\Phi(d) = \int_{x=d} F(x,y)\,\mathrm{d}s \;= \int_{-m}^{m} \phi\left(\sqrt{d^2+y^2}\right)\mathrm{d}y, \tag{5.4.3}$$

which in full is

$$\Phi(d) = \int_{-m}^{m} s\left((u-1)\exp\left(-\left(\frac{\sqrt{d^2+y^2}-\mu}{2\sigma}\right)^2\right)+u\right)\mathrm{d}y. \tag{5.4.4}$$

No closed form solution could be found for this expression. Expanding the function via a Taylor series gives an integrable result, however it requires many terms to give a close approximation, and the result is extremely sensitive to the expansion point used during this process. Integrating the result further increases the instability. It might be possible to tune the expansion point to give good results for specific data, however a general solution would still be preferred, and so a method for numerically approximating the integral was developed. Assume the values for $\phi(d)$ are known for k discrete, equally spaced values of d, $\mathbf{d}_s = (0, n, 2n, \ldots, (k-1)n)$. Then, create a $2k \times k$ array D, where

$$D_{i,j} = \begin{cases} \phi(\|(i,j)-(k,1)\|n) & \text{if } i < k \\ \phi(\|(i,j)-(k+1,1)\|n) & \text{otherwise} \end{cases} \quad \text{for } i = 1,\ldots,2k,\ \ j = 1,\ldots,k \tag{5.4.5}$$

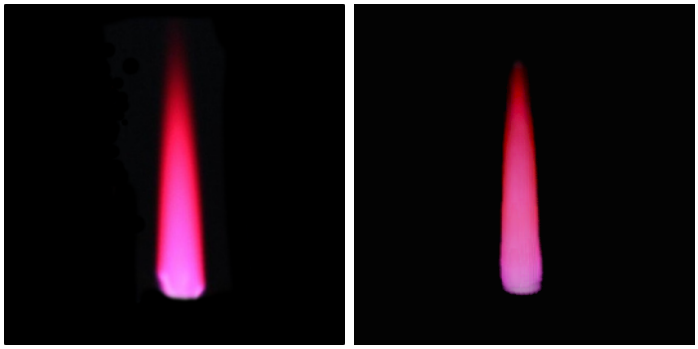


Figure 5-15: Original image (left), rendering of the fitted flame (right)

Values for $\phi(d)$ where $d \notin \mathbf{d}_s$ can be found by interpolating the known values. Then

$$\Phi(d) = \frac{\sum_{i=1}^{2k} D_{i,d}}{k} \tag{5.4.6}$$

This process for converting $\phi(d)$ into $\Phi(d)$ can be incorporated into the Levenberg-Marquardt algorithm, allowing the regular density function to be fit to the points (d, δ) taken from the two-dimensional image. This gives sensible values for the parameters μ, σ, u, m, and s, that when projected into two dimensions, give a result like the data measured. This is a brute force solution to the problem, which understandably slows down the algorithm, but still to an acceptable level: about 25 seconds to find parameters for a two-dimensional lighter image, compared to about 15 seconds for the three-dimensional version.

Results of the two-dimensional fitting process are given in Figures 5-15 and 5-16. The fitted parameters for the core are just (x, z) points. For these images, a fixed depth of $y = 0.5$ was used, so the core is planar. The second figure shows that the two-dimensional reconstructions look accurate from the same angle as the input images, and like realistic flames from the side – however they do not match the three-dimensional versions due to this. Interpreting a varying depth of the core from the shape of the flame in two dimensions is another interesting area for future work. This might be achieved by assuming the volume of the flame is preserved between frames, or based on priors learned from three-dimensional flames.

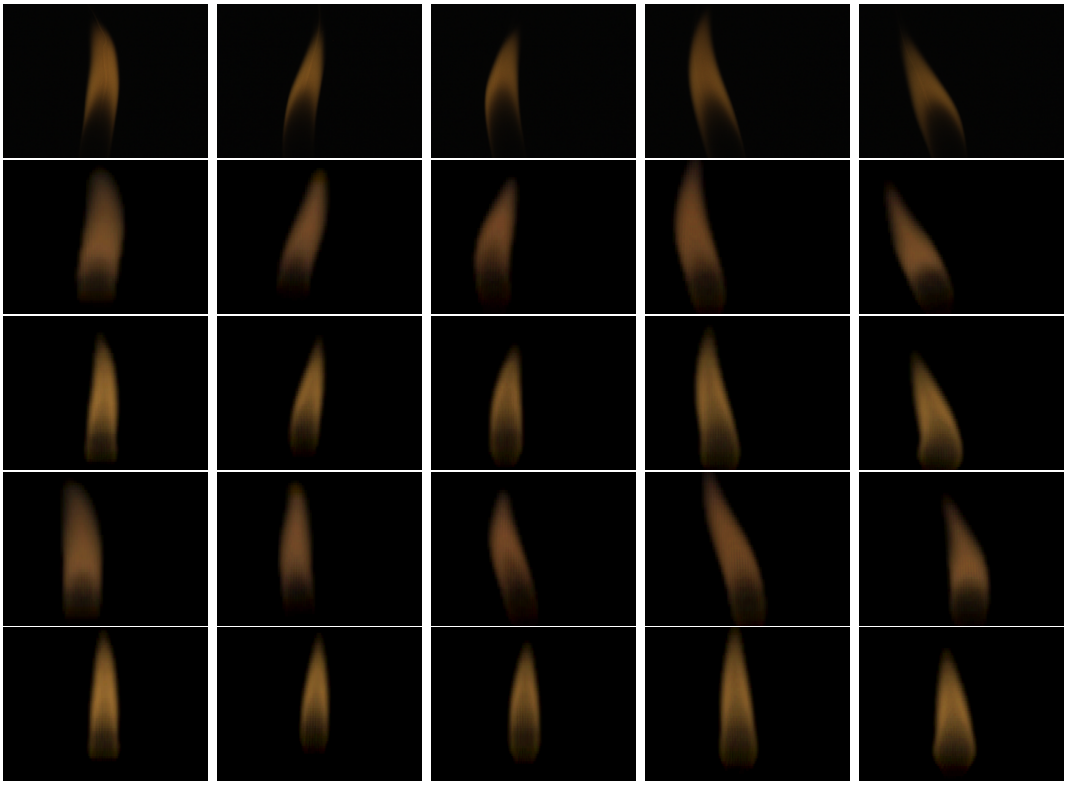


Figure 5-16: Rows 1,2,3: original images, 3D fit, 2D fit
Rows 4 and 5: 3D and 2D fits rotated 90°

5.5 Conclusion

This chapter completes the original hypothesis and objective: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* In Chapter 3 the acquisition of flame videos and the reconstructing of voxel models was documented, and in Chapter 4 it was shown that a flame core representation produced a realistic result, and was intuitively editable. This chapter gives a method of fitting these models to the real video data in voxel format, hence giving a full pipeline that can start with real video, and end with a visually intuitively editable three-dimensional flame model. The results are convincing as flames, and adequately similar to the original video, especially when the flame is in motion.

Furthermore, a simple technique for finding parameters that fit to two-dimensional, uncalibrated input was presented. While the method is not able to reconstruct any depth information from this, it can make a good estimation of the density parameters. This

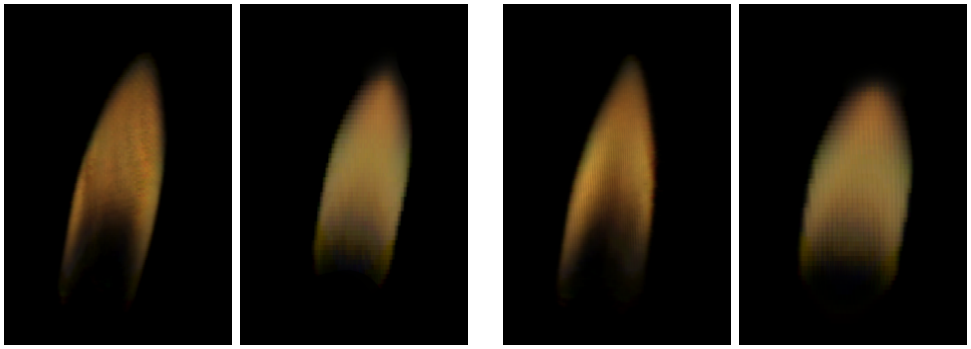


Figure 5-17: Two views of the same flame; the model looks correct on the left, but is too wide on the right

means that they can be applied to other flames that are fully three-dimensional, to create brand new three-dimensional flames. One interesting area of future work would be to use volume preservation between frames, or prior information learned from the three-dimensional data, to infer depth from a single two-dimensional image.

One shortcoming of the flame core model becomes clear when it is compared to its original three-dimensional data, and that is that not all flames are radially symmetric. As highlighted in Figure 5-17, some flames are narrower in one direction than they are in the other. The model expects them to be roughly symmetrical, so that the density from the centre of the core does not change based on the angle. This can cause an issue in fitting; since the data is averaged in terms of distance from the core, this can result in a bimodal distribution shape appearing – one for the further peak, and one for the closer one. This is visible in Figure 5-18. The solution used was to approximate between the two, however this leads to the flame appearing too wide from some angles compared to the original data, and sometimes too large a σ value is found, which makes the flame seem blurry.

A possible solution for this would be to fit a more complicated model for the density component. For instance, rather than the density being seen as a curve $d(u)$, it could be a surface $d(u, \theta)$ where u is a point along the core, as usual, and θ is an angle from some fixed vector (possibly the binormal to the core). This could represent any asymmetric flame, while still using the same density function to represent the shape. However, it would complicate the model considerably and possibly cause issues with the amount of training data available. Since the resulting flames are realistic and still have the appearance of the target flame, this was considered acceptable.

Some methods for fitting to more complicated flames with more than one lick have been

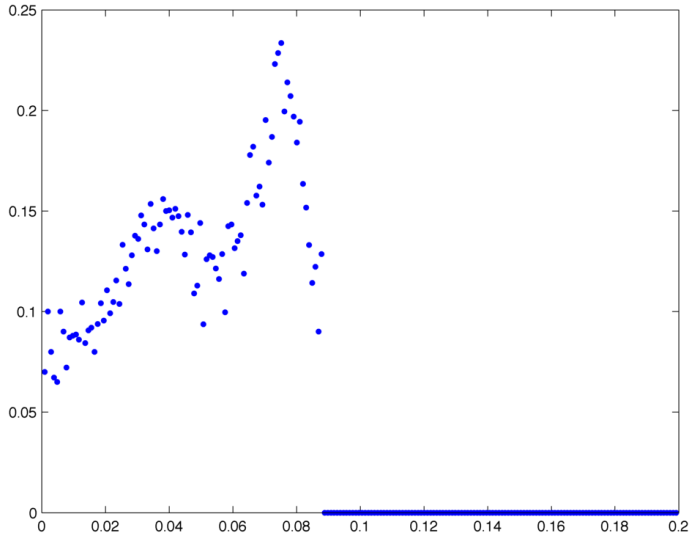


Figure 5-18: The asymmetry causes a bimodal shape in the density

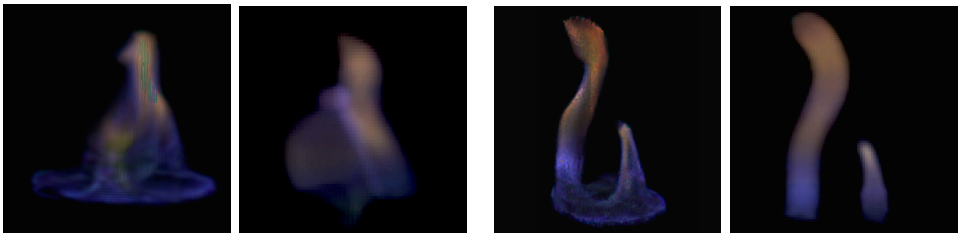


Figure 5-19: Examples of fit with multiple licks

given. Unfortunately, as addressed in Chapter 3, the only dataset with complicated flames has some problems with motion blur, and so a good fit could not be found for many flames. This makes it hard to assess since a moving version could not be generated. Two example flames with two licks are shown in Figure 5-19, one where the licks are merging and one where they are separate. This dataset was not the focus since much better results were obtained from the others, it is possible better results could be achieved.

5.5.1 Evaluation

Two comparisons are possible to numerically evaluate the goodness of fit of the flame models. First, the models can be compared to the three-dimensional reconstructions on which they are based; in other words, evaluating the objective equation in Equation 5.1.1. Since the model is being applied to different data, it is necessary to normalise the error

	Mean	Standard deviation
Candle	0.608	0.0909
Lighter	0.636	0.0847
Alcohol	0.924	0.1998

Figure 5-20: Table showing NRMSE_{3D} (Equation 5.5.1) between volumes and renderings

values in order to give a fairer comparison, both between datasets and between frames. Figure 5-20 shows the average errors produced when fitting to the three datasets, using the normalised root mean squared error, calculated as follows:

$$\text{NRMSE}_{3D} = \sqrt{\frac{\sum_{x,y,z=1}^{N} \left(\varphi(x,y,z) - \phi(\delta,\mu,\sigma,u,m,s)\right)^2}{\sum_{x,y,z=1}^{N} \varphi(x,y,z)^2}} \tag{5.5.1}$$

where the terms are defined as in Section 1.3.1. The error is normalised by the sum of the original voxel densities squared. As the table shows, the model fits roughly as well to the candle dataset as is does to the lighter dataset. As expected, the fit to the alcohol dataset is much worse. The flame shown in Figure 5-17 has an error score of 0.713, which is just over one standard deviation higher than the mean. This is consistent with the theory that the radial symmetry problem accounts for the flames with the higher error values.

Another point of comparison is to use the S matrix from the image-based tomographic method [IM04] (described in Sections 2.4.2 and 3.4) to reproject the rendered and volumetric flames into the original camera conditions, and compare these to the originals. This enables an evaluation of the flame core method as a model of the original data, in comparison to the tomographic method. These values are given in Figure 5-21, and are the average two-dimensional normalised root mean squared error between the original image and the reprojected image:

$$\text{NRMSE}_{2D} = \sqrt{\frac{\sum_{x,y=1}^{N} \left(o(x,y) - r(x,y)\right)^2}{\sum_{x,y=1}^{N} o(x,y)^2}} \tag{5.5.2}$$

where o is the original image and r is the image formed by projecting the voxel reconstruction into the exact same camera conditions, using the matrix S that was formed during the tomographic process and used to reconstruct the three-dimensional volumes. As expected, the values show that the flame core model performs significantly worse for the alcohol dataset than the candle and lighter, whereas the tomographic method gives a similar performance across all three. Naturally the tomographic method performs better

	Tomography [IM04]		Flame Core	
($\times 10^{-3}$)	Mean	Standard deviation	Mean	Standard deviation
Candle	0.356	0.0230	0.426	0.0360
Lighter	0.486	0.0428	0.605	0.0488
Alcohol	0.490	0.0991	1.526	0.2196

Figure 5-21: Table showing NRMSE_{2D} (Equation 5.5.2) between original images and re-projected renderings

overall, since the flame core models were fit to its results – a better performance against the original images would have to be coincidental. However, the flame core method still produces comparatively good results on the candle and lighter datasets, and offers intuitive control over its appearance, which the tomographic method does not.

The only other method which creates three-dimensional results and uses real video as input is flame-sheet decomposition [HK03, HK07] (see Section 2.4.1). Unfortunately no implementation or density field results were available to compare directly. However, the authors have published some images of their method applied to the alcohol dataset. The method ensures that the reconstructions are numerically consistent with the input views, which results in a clean image which can show less noise than the tomographic densities when rendered in two-dimensions. However, it does not ensure spatial coherence, which can lead to parts of the density floating in mid-air between two licks when rendered from novel angles. Furthermore, the solution is inherently two-dimensional, and so view interpolation on the vertical axis would reveal the flames to look flat, or a collection of slices. The three methods applied to a single alcohol flame are shown in Figure 5-22. The flame core result is clearly the furthest from the true flame, which is expected since the method does not work well on this dataset, as previously mentioned. On the other datasets, the results would be much closer. Between the two techniques, the flame-sheet method would probably still have the more visually accurate result from the original camera angles or when interpolating between them, but the flame core method does have the advantage of being a true three-dimensional density. As such, it will look solid from above and will not create any disjointed areas of floating density (these are visible in the middle images in Figure 5-22). Furthermore, it is the only one of the three that offers control over its appearance.

The models representing each frame in the animation can be edited in a visually intuitive way, and these modifications can be applied in bulk to an entire animation. In the next chapter, another approach is explored, which is to generate new unique animations that

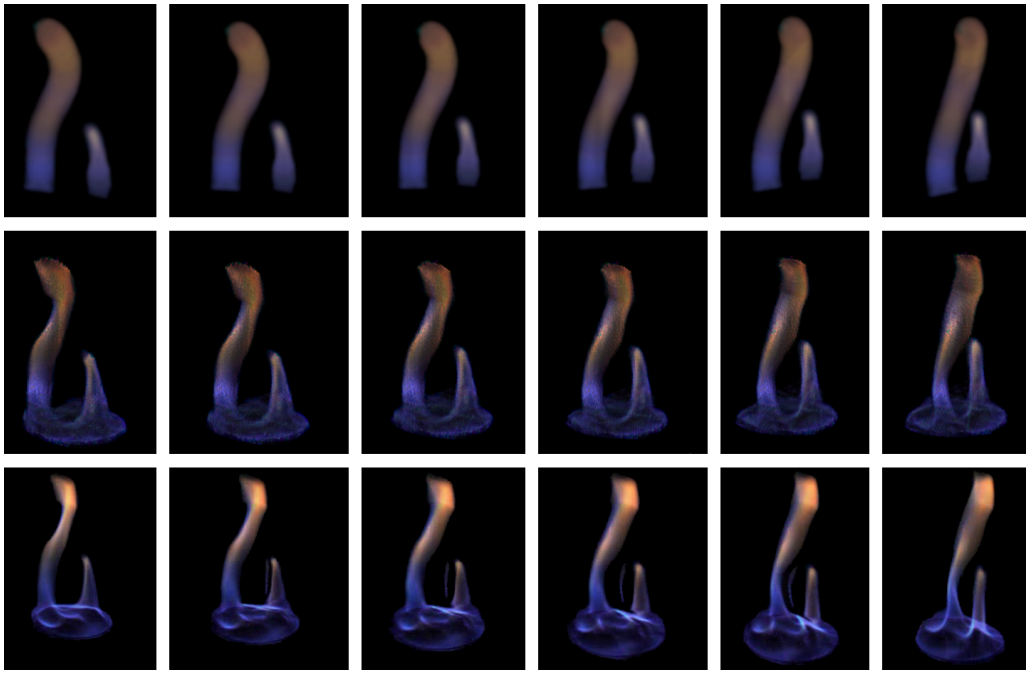


Figure 5-22: From top to bottom: flame core method, image-based tomography [IM04], flame-sheet decomposition [HK03]

look similar to existing ones. In other words, create arbitrarily long sequences of flames could be generated from a pre-existing video. Now, the original video itself is the method of control – there is no need for an artist to create the animation frame by frame. To generate new video of a candle, simply provide an existing video of a candle. Furthermore, new controls will be established on top of this, to allow specific deviation from the original video.

Chapter 6
Generative Model of Flames

This chapter's contribution is a generative model for creating novel animations of flames from a pre-existing sequence. These animations look visually similar to the training data, but are distinct and of unconstrained length. Further, the output can be controlled in a way that creates completely distinct flames, that still move in a realistic way determined by the underlying training.

6.1 Overview

Once an entire sequence of flames has been created, most likely from an input video (Chapter 5), the evolution of the parameters through this sequence can be used to create new unique animated flames. Motion has not been an explicit element of the flame core model up until this point. This is because a sequence of flames that create an animation, much like the images that make a normal video, are sufficient to create the illusion of motion. For simple flames with just a single core, this progression from one frame to the next forms the base for the generation process.

The generative method itself, the first contribution of the chapter, is similar to and was inspired by previous work on Chaotic Modelling (Section 2.3.4) and Video Textures (Section 2.3.1). It works over the parameter spaces of the flame core model, rather than directly on the input data (e.g. images or human motion). Each parameter from the training data forms a time series in a high dimensional deflated space. New states are found by first following the motion of nearby points from the training, and then adding some noise to create a small variation.

For complex flames that consist of multiple licks per frame, a simple time series is not

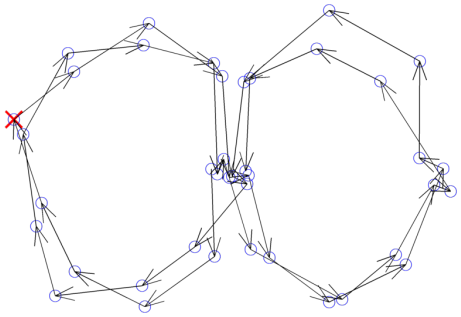
sufficient to encapsulate the motion. Some idea of how the licks move, split, or merge, is also required. This would also require a modification of the generative process. Some work was done to extract this motion, however the complex source data was not of high enough quality to produce good quality animations, so the results cannot be tested. With higher quality data this would be a logical area of future work.

Finally some methods of control over the generation process are discussed. This demonstrates that from a single input video, it is possible to create a range of different looking flames, the second contribution of this chapter.

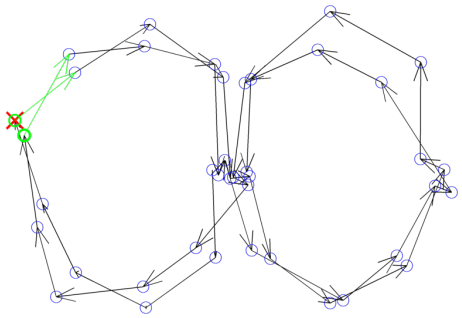
6.2 Generative Model

The first contribution of this chapter, and one of the overall main contributions of this thesis, is the ability to generate unique animated flames which look similar to an existing sequence. This approach uses a collection of flames in the flame core format, henceforth referred to as the training data. The general idea is to represent each parameter in a high dimensional space and trace out a path, adding some random variation. Parameters are initially considered completely independently, however this can produce odd results. A slight modification allowing multiple parameters to be considered at once significantly improves this.

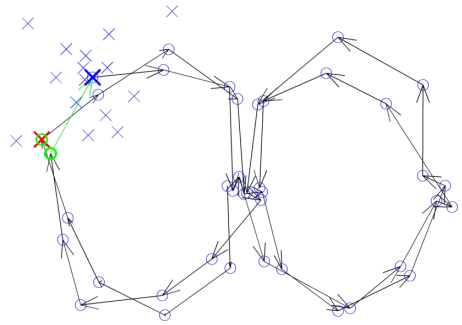
In slightly more detail, an overview of the generative process for each parameter (or collection of parameters) is as follows. First, the training data is converted into a time series $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M)$, where $\mathbf{x}_i \in \mathbb{R}^N$. Principal component analysis [Jol02] is used to reduce the dimensionality, forming an eigenbasis of deflated points. The generation process then follows these steps:



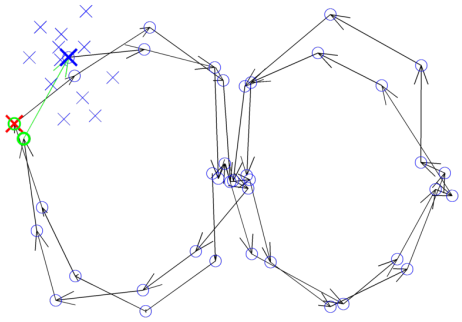
The training data (circles and arrows) form a path. Pick a starting point from the training, shown here as a red cross.



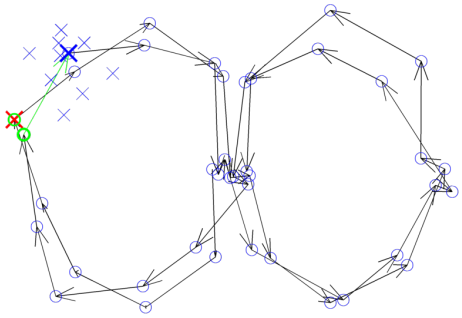
Find nearby points from the training, shown here as green circles. Then find the subsequent points, shown as green arrows.



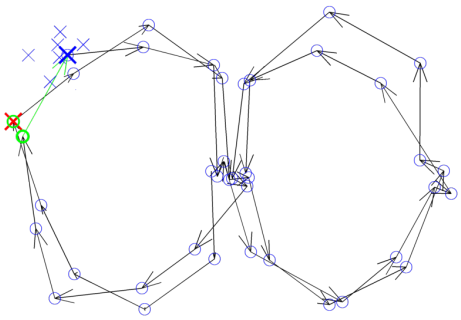
Pick one of these subsequent points, shown here as the bold blue cross. Then generate points around this using a multivariate normal, shown here as smaller blue crosses.



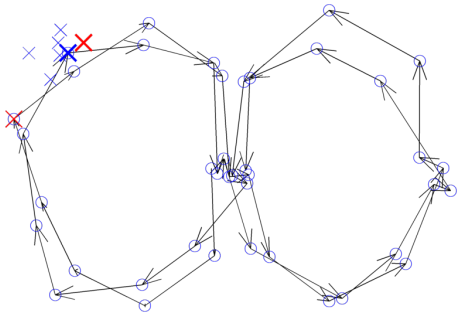
The points are filtered based on three conditions: extreme values are filtered using the multivariate normal likelihood;



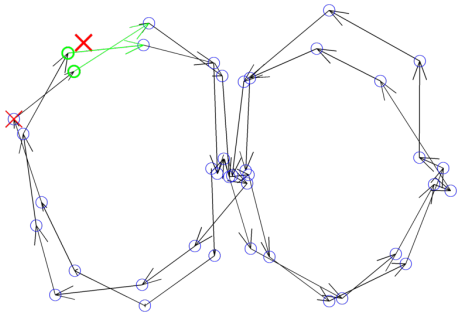
kernel density estimation filters points that lie off the path;



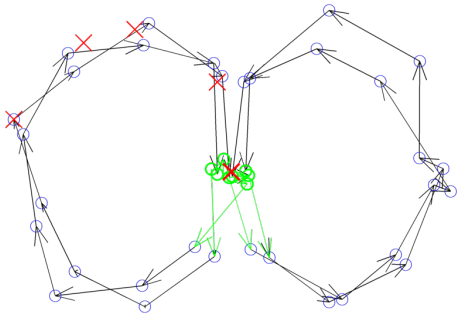
and a von Mises-Fisher distribution is used to filter points that vary too much in direction from the previous (for frames $t > 1$).



Pick one of these points, this is the next point in the animation.



The process now repeats.



Note that when paths cross, the animation will randomly pick routes that may diverge from the original.

A more formal description of the method is presented in three sections: initial processing of the training data, the loop that generates new frames, and then the modifications that improve the result.

6.2.1 Pre-processing

In this and the next section, each parameter of the model is considered independently. Later some parameters will be considered together, but the actual generation method is the same. It is assumed that there exists a sequence of simple flames in the flame core format: $(\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \dots)$ where $\mathcal{F}_i$ is a flame which has exactly one lick, comprised of a core and the corresponding density function. Therefore, this sequence can be written $((c_1, d_1), (c_2, d_2), \dots)$, where as in Chapter 4,

$$c : [0,1] \to (x, y, z) \in [0,1]^3 \tag{6.2.1}$$

$$d : [0,1] \to (\mu, \sigma, u, m, s) \in \mathbb{R}^5, \ \sigma \neq 0. \tag{6.2.2}$$

Since this is a simple flame with just one lick per frame, it is assumed that the sequence describes the motion, i.e. flame $\mathcal{F}_{t+1}$ directly follows flame $\mathcal{F}_t$. Complicated flames will be considered in a future section.

First each parameter of the training data is converted into a time series representation. The core of the lick $c(u) = (x(u), y(u), z(u))$ is considered one parameter, as are each of the five density parameters $d(u) = (\mu(u), \sigma(u), u(u), m(u), s(u))$. No specific form is imposed on these functions, they might be cubics, piecewise polynomials, Bézier splines, or something completely different. Instead, discrete forms of these parameters from sampling along the core are used. For example, sampling the density function at N equally spaced points along the core produces several vectors, one of which is $\boldsymbol{\mu} = (\mu_1, \dots, \mu_N)$ where μ_1 gives the value corresponding to $d(0)$ and μ_N to $d(1)$ – the base and tip of the lick respectively. For the parameters which produce vectors at a given point on the core (specifically: the core c, and in colour flames, the scale s and uniform ratio u), a one dimensional vector should be formed with any invertible mapping. For example, a single lick can give the following core vector

$$\mathbf{c} = (c_1^x, c_1^y, c_1^z, c_2^x, c_2^y, c_2^z, c_3^x, \dots). \tag{6.2.3}$$

This is done for every flame in the training set, giving a time series for each parameter. For example, the time series of cores is

$$\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_M) \tag{6.2.4}$$

where M is the number of flames in the training data.

In total, there will be six time series: one for the core, and one for each density parameter.

The following processes are the same for each parameter. Therefore, they will be described for the placeholder variable

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M),\ \mathbf{x}_i \in \mathbb{R}^N \tag{6.2.5}$$

where N is the number of sample points multiplied by the parameters per sample (1 or 3).

Next principal component analysis (comprehensively covered in [AW10]) is used on the training data, forming an eigenbasis which optimally represents the correlation between the points. The mean point:

$$\boldsymbol{\mu} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i \tag{6.2.6}$$

is the origin of the eigenbasis, where M is the number of points in the training data. The eigenvectors U and eigenvalues D are given by

$$UDU^{\mathsf{T}} = \frac{1}{M} XX^{\mathsf{T}} - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}. \tag{6.2.7}$$

These points will lie on a manifold, which will typically have lower intrinsic dimension than the space, and will later be locally estimated to find new reasonable points. The benefit of this process is that the low energy components, i.e. the eigenvectors with a low eigenvalues, can be removed. This deflation process reduces the dimensionality, which improves performance, and can remove noise. So, just using the n eigenvectors corresponding to the largest eigenvalues, and calling these U_n, then

$$\mathbf{x}_i \mapsto {U_n}^{\mathsf{T}}(\mathbf{x}_i - \boldsymbol{\mu}) \tag{6.2.8}$$

maps from the training data to the eigenbasis.

Use the modified eigenvectors U_n to project the training data into this deflated eigenbasis, for each parameter. This new time series of eigenbasis points $\mathbf{X}^e$ will be used to create the animation.

6.2.2 Animation

The animation goal is the following: given a current state for each parameter, find the next state. This, coupled with an initial state, generates a novel animation. The aim

is to approximate some underlying function f that predicts the next state, based on the previous states and the training data

$$\mathbf{y}_{t+1} = f((\mathbf{y}_t, \mathbf{y}_{t-1}, \ldots, \mathbf{y}_1), \mathbf{X}^e) \tag{6.2.9}$$

If in general $f((\mathbf{y}_t, \mathbf{y}_{t-1}, \ldots, \mathbf{y}_1), \mathbf{X}^e) = f((\mathbf{y}_t), \mathbf{X}^e)$, then this process is memoryless, it has the Markov property. Some possibilities of the function f that hold this property are discussed in [BS09]. In the method presented here, like the prior work on video textures [SSSE00], $f((\mathbf{y}_t, \mathbf{y}_{t-1}, \ldots, \mathbf{y}_1), \mathbf{X}^e) = f((\mathbf{y}_t, \mathbf{y}_{t-1}), \mathbf{X}^e)$. i.e. the next estimate depends on the previous two in the series, rather than just one. This could be called the second-order Markov property, if the original is considered first-order. That is, the velocity obeys the regular property.

The approximation of f has two parts: motion estimate, followed by randomisation.

Motion Estimate

For a given point $\mathbf{y}_t$, find the nearest neighbours from the training data, within a distance d which is set to the mean distance between points in the training

$$\mathbf{NN} = \{\mathbf{x} \in \mathbf{X}^e \text{ such that } \|\mathbf{y}_t - \mathbf{x}_i\| < d\} \tag{6.2.10}$$

Then each neighbour $\mathbf{x}_i$ has a corresponding next point $\mathbf{x}_{i+1}$ in the training data, call these

$$\mathbf{NN}^+ = \{\mathbf{x}_{i+1} \text{ such that } \mathbf{x}_i \in \mathbf{NN}\} \tag{6.2.11}$$

One of these will be chosen for the motion estimate for the point $\mathbf{y}_t$. Specifically, a random element $\mathbf{x}$ is chosen from $\mathbf{NN}^+$ according to the probability mass function $f_{\mathbf{x}}$, which uses a von Mises-Fisher distribution to measure whether the next point will move in a similar direction to the direction between the previous two points

$$f_{\mathbf{x}}(\mathbf{z}) = \frac{1}{\omega} \text{vMF}\left(\frac{\mathbf{z} - \mathbf{y}_t}{\|\mathbf{z} - \mathbf{y}_t\|} \middle| \frac{\mathbf{y}_t - \mathbf{y}_{t-1}}{\|\mathbf{y}_t - \mathbf{y}_{t-1}\|}, \kappa\right) \text{ for all } \mathbf{z} \in \mathbf{NN}^+ \tag{6.2.12}$$

where $\text{vMF}(\mathbf{a}|\boldsymbol{\mu}, \kappa)$ is the d-dimensional von Mises-Fisher probability density function

$$\text{vMF}(\mathbf{a}|\boldsymbol{\mu}, \kappa) = C_d(\kappa) \exp\left(\kappa \boldsymbol{\mu}^\mathsf{T} \mathbf{a}\right). \tag{6.2.13}$$

The normalisation constant $C_d(\kappa)$ can be found in the literature, such as [DS03a, BDG$^+$05]. It is not important here because these values are normalised by ω,

$$\omega = \sum_{\mathbf{z} \in \mathbf{NN}^+} \text{vMF}\left(\frac{\mathbf{z} - \mathbf{y}_t}{\|\mathbf{z} - \mathbf{y}_t\|} \middle| \frac{\mathbf{y}_t - \mathbf{y}_{t-1}}{\|\mathbf{y}_t - \mathbf{y}_{t-1}\|}, \kappa\right). \tag{6.2.14}$$

κ controls the concentration of the distribution, and can be fitted to the training data, or simply set to a constant, such as $\kappa = 10$.

Obviously this is unsuitable for the first frame of the animation when $\mathbf{y}_{t-1}$ does not exist. In this case, $f_{\mathbf{x}}$ uses the distance from the corresponding nearest neighbour points to select the random element

$$f_{\mathbf{x}}(\mathbf{z}_i) = \frac{1}{\omega'} \exp\left(-\frac{\|\mathbf{z}_{i-1} - \mathbf{y}_t\|}{d}\right) \text{ for all } \mathbf{z}_i \in \mathbf{NN}^+. \tag{6.2.15}$$

Note that each $\mathbf{z}_i \in \mathbf{NN}^+$ corresponds to one $\mathbf{z}_{i-1} \in \mathbf{NN}$. d is the mean distance between points, as before, and ω' normalises these values

$$\omega' = \sum_{\mathbf{z} \in \mathbf{NN}} \exp\left(-\frac{\|\mathbf{z} - \mathbf{y}_t\|}{d}\right) \tag{6.2.16}$$

The random element $\mathbf{x} \in \mathbf{NN}^+$ is the point that estimates the motion of the parameter at this point in the animation. However, this point is from the training data. To create a similar value, a nearby random point is chosen in the next step.

Randomisation

To randomise this parameter, a nearby point will be similar but meaningfully different. The point will be chosen based on kernel density estimation with the training data $\mathbf{X}^e$. This means the point will lie on, or very close to, the manifold created by the training points. Provided this is not the first new frame, the von Mises-Fisher weighting used in the motion estimate section will be used as well. This means that any generated point will still be going in roughly the correct direction.

A number of points are generated from a multivariate normal distribution centred around

the point from the training data:

$$\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_k) \sim \mathcal{N}(\mathbf{x}, \Sigma). \tag{6.2.17}$$

Σ is a diagonal matrix that has the mean distance between subsequent points from the training data along the diagonal. This is to ensure that the randomisation does not move further away from the original point than the average distance between two points, to ensure visual smoothness. It can be scaled by a user parameter to increase or decrease the appearance of erratic movement.

Then a random element $\mathbf{r}$ from $\mathbf{R}$ is chosen according to the probability mass function $f_{\mathbf{r}}$, given by

$$f_{\mathbf{r}}(\mathbf{z}) = \frac{1}{\omega^*} \operatorname{MVN}(\mathbf{z}|\mathbf{x}, \Sigma)\ \operatorname{KDE}(\mathbf{z}|\mathbf{X}^e)\ \operatorname{vMF}\left(\frac{\mathbf{z}-\mathbf{y}_t}{\|\mathbf{z}-\mathbf{y}_t\|}\middle|\frac{\mathbf{y}_t-\mathbf{y}_{t-1}}{\|\mathbf{y}_t-\mathbf{y}_{t-1}\|}, \kappa\right) \text{ for all } \mathbf{z} \in \mathbf{R} \tag{6.2.18}$$

where $\operatorname{MVN}(\mathbf{a}|\boldsymbol{\mu}, \Sigma)$ gives the usual probability density function of the multivariate normal distribution, so this gives a low probability to extreme values. $\operatorname{vMF}(\mathbf{a}|\boldsymbol{\mu}, \kappa)$ is the von Mises-Fisher probability density function as in Equation 6.2.13, so this weights towards points which continue moving the same direction. Finally the kernel density estimation $\operatorname{KDE}(\mathbf{a}|\mathbf{S})$ is

$$\operatorname{KDE}(\mathbf{a}|\mathbf{S}) = \frac{1}{n}\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{a} - \mathbf{s}_i), \tag{6.2.19}$$

this weights towards points which lie around the existing points, i.e. on the manifold. A Gaussian kernel is used, with the bandwidth given by Silverman's rule of thumb [Sil86], where if d is the dimension of the deflated eigenbasis

$$\sqrt{\mathbf{H}_{ii}} = \left(\frac{4}{d+2}\right)^{\frac{1}{d+4}} n^{\frac{-1}{d+4}} \sigma_i. \tag{6.2.20}$$

For training data with more erratic motion, a more accurate kernel density estimator may be required. One suitable example is Kristan et al.'s work [KLS11], for which an implementation is publicly available. However this increases the computation time significantly when finding the kernels. Finally, ω^* normalises these terms, as before

$$\omega^* = \sum_{\mathbf{z}\in\mathbf{R}} \operatorname{MVN}(\mathbf{z}|\mathbf{x}, \Sigma)\ \operatorname{KDE}(\mathbf{z}|\mathbf{X}^e)\ \operatorname{vMF}\left(\frac{\mathbf{z}-\mathbf{y}_t}{\|\mathbf{z}-\mathbf{y}_t\|}\middle|\frac{\mathbf{y}_t-\mathbf{y}_{t-1}}{\|\mathbf{y}_t-\mathbf{y}_{t-1}\|}, \kappa\right) \tag{6.2.21}$$

This random element $\mathbf{r}$ is the next point in the animation for this parameter

$$\mathbf{y}_{t+1} = \mathbf{r} \tag{6.2.22}$$

so this concludes the generative process. The parameter is then unprojected from the eigenbasis using the inverse of the map in Equation 6.2.8. Once the two steps, motion estimate and randomisation, have been completed for each parameter, the unprojected parameters form the next flame in the animation. This can be rendered in the usual way, or just stored to form the new sequence.

6.2.3 Parameter Modifications and Concatenation

In general, updating each parameter in the model independently is beneficial. It means that even if the randomisation step were skipped, the random element in the motion estimate will lead to combinations of parameters that are unique. However, in real flames it was found that the μ, σ, and m parameters were not independent. As a reminder, these refer to the distance of the peak and standard deviation of the density, and the maximum distance (a cut off distance) respectively.

First, the m parameter depends partly on μ and σ. This is reasonable: a wide portion of flame might have a high μ, low σ, and high m, a narrow flame might be the reverse, in particular a low m. However, it would be incorrect to generate a flame with high μ, low σ, and low m – this would cut out the density at a different point from either flame, way before the peak.

Instead, the values of m are modified to represent the Mahalanobis Distance [Mah36], i.e. the number of standard deviations away from the mean

$$m' = \frac{m - \mu}{\sigma}. \tag{6.2.23}$$

Now, both the narrow and wide slice of flame could have the same value for m'.

It was also found that certain randomised combinations of these parameters produced displeasing results. This was only in a limited number of cases, however when generating several hundred frames to form a video, it became apparent often. One hypothesis for this is simply that some combinations of μ and σ, for example, simply do not occur in realistic flames, but independently the values do exist. To solve this problem, the joint space is modelled, by simply concatenating the eigenbasis vectors.

This process was as follows, create the eigenbases for each parameter, and project the points into these bases ($\mathbf{X}_\mu, \mathbf{X}_\sigma, \mathbf{X}_{m'}$). Concatenate each point, forming

$$\mathbf{X} = ((\mathbf{x}_\mu^\mathsf{T}, \mathbf{x}_\sigma^\mathsf{T}, \mathbf{x}_{m'}^\mathsf{T})^\mathsf{T}) \tag{6.2.24}$$

This data is whitened: the standard deviations of each dimension are normalised, so that the kernel density estimation does not favour one parameter at the expense of another. In other words, the i^{th} element of $\mathbf{X}^\text{W}$ is given by

$$\mathbf{x}_{ij}^\text{W} = \frac{\mathbf{x}_{ij}}{\sigma_j} \text{ for } i = 1, \ldots, N \text{ and } j = 1, \ldots, k \tag{6.2.25}$$

where N is the number of points in the training set, k is the combined dimensionality of the points from the eigenbasis representation, and

$$\boldsymbol{\sigma} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})^2} \tag{6.2.26}$$

This new set of points is treated as a brand new parameter in the generation process, from the beginning. That is: an eigenbasis is formed from these points, this is followed by motion estimation, randomisation, and then unprojection from the eigenbasis – as in the previous sections. This results in a single vector that is a combination of the three new parameters μ, σ, and m. The reverse of the whiten operation is performed: each dimension is multiplied by the original standard deviation. Three vectors are extracted, one for each parameter, according to their original dimensionality, and then these are unprojected from their respective eigenbases, to give the final generated parameters.

A comparison of some results before and after the concatenation process is shown in Figure 6-1. These flames were generated based on the candle dataset. Video of a sequence of generated flames can be found on the accompanying disc.

The implementation of the generation algorithm takes about 13 seconds per frame (not including rendering). This is mostly caused by a simple implementation of the kernel density estimation in Equation 6.2.19. Bearing in mind the other parts, a more efficient version could easily bring the overall time down to around 1 second per frame. If the motion estimates were precomputed, it might be possible to achieve real-time results.

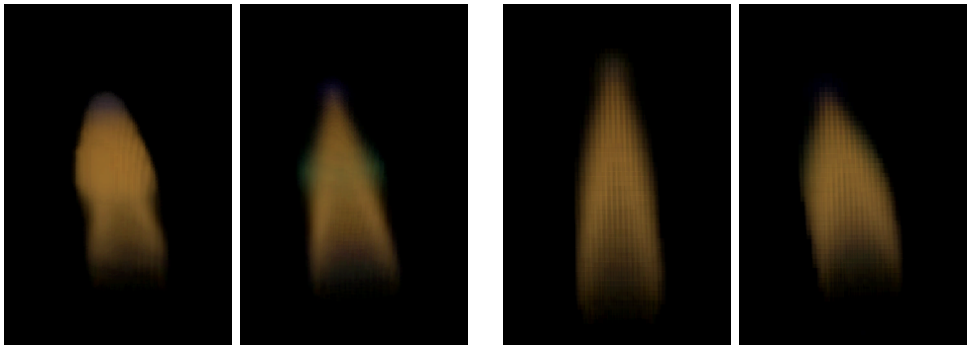


Figure 6-1: Flames before (left) and after (right) μ, σ, and m were concatenated

6.3 Complex Motion

The previous section assumed that the input flames were simple, that they just had one lick per flame. This obviously doesn't account for flames which have multiple licks, but it also cannot handle missing frames from the training data. Rather than inferring motion between subsequent elements in the time series, a more sophisticated motion model was designed. Unfortunately, the complex flame data was not dense enough to create an animation using this data, so the results of matching licks could not be tested in animations. The data limitations are discussed in Chapter 3. This more sophisticated model is presented below, as it has some limited application to the regular generation process, and may lead to interesting future work.

A lick of flame can either move, merge, split, or become extinguished. This can be represented by a collection of pairs, each one containing two licks, which indicates that these are a match, i.e. the first evolves into the second. Therefore if the first flame has two licks, $\{\mathcal{L}_a, \mathcal{L}_b\}$, and it evolves into a flame which also has two licks $\{\mathcal{L}_1, \mathcal{L}_2\}$ then a possible matching could be:

$$\mathbf{M}^{\text{example}} = \{(\mathcal{L}_a, \mathcal{L}_1), (\mathcal{L}_b, \emptyset), (\mathcal{L}_a, \mathcal{L}_2)\}. \tag{6.3.1}$$

This reads that the first lick in the first flame split into the first and second licks in the second flame, while the second lick in the first flame died out. This matching is illustrated in Figure 6-2. This notation also allows for licks merging together. Note that if a matching is not present, it is assumed to die out, so this is equivalent:

$$\mathbf{M}^{\text{example}} = \{(\mathcal{L}_a, \mathcal{L}_1), (\mathcal{L}_a, \mathcal{L}_2).\} \tag{6.3.2}$$

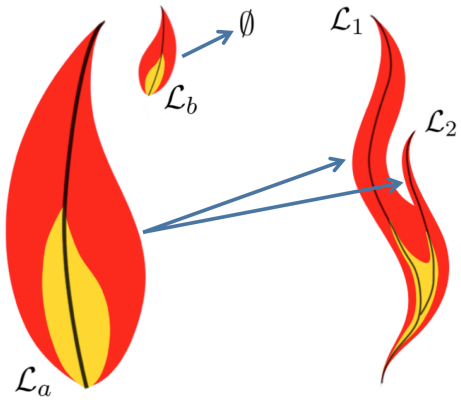


Figure 6-2: A conceptual example of the motion estimate between frames

The simple flame motion from the previous section, using a time series, is just a subset of this pair-based matching. The new system has the advantage of being able to create discontinuities in the motion, for example if the source flame moves out of frame, or the fitting is bad for some other reason. The generation process would be extremely similar in this case. In the motion estimate, whenever the algorithm uses the point $\mathbf{z}_{i+1}$ to correspond to the next point of $\mathbf{z}_i$, instead it should find a pair of elements $(\mathbf{z}_i, \mathbf{z}_j)$ in the matching, and use $\mathbf{z}_j$.

For complex flame motion, the generation process could change significantly. If a point that splits into two is chosen in the motion estimate, then the generation process should do the same, and then each lick should be processed independently, and continue to evolve with its own parameters, until the flames merge or one dies out. While this was not implemented due to the lack of training data, a method for finding how licks match between two flames was developed. This is presented next.

6.3.1 Motion of Complicated Flames

For this process the original three-dimensional data in point cloud format is used, divided into individual licks, as in Section 5.2.2. In other words, consider a pair of flames adjacent in time, flame $\mathcal{F}_A$ and $\mathcal{F}_B$, with corresponding licks $\{\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \ldots, \mathbf{A}_n\}$ and $\{\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \ldots, \mathbf{B}_m\}$. Here each lick is a collection of points in three-dimensional space, and each lick corresponds to exactly one core and density function.

The matching is based on the Kullback-Leibler divergence of the latter lick from the

former. This divergence, $D_{\mathrm{KL}}(\mathcal{P}\|\mathcal{Q})$, is a measure of the information lost when $\mathcal{Q}$ is used to approximate $\mathcal{P}$. Specifically the licks are modelled as multivariate normal distributions, and the divergence is calculated as follows:

$$D_{\mathrm{KL}}(\mathcal{N}_{\mathbf{A}}\|\mathcal{N}_{\mathbf{B}}) = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_{\mathbf{B}}^{-1}\Sigma_{\mathbf{A}}\right) + (\mu_{\mathbf{B}} - \mu_{\mathbf{A}})^{\mathsf{T}}\,\Sigma_{\mathbf{B}}^{-1}(\mu_{\mathbf{B}} - \mu_{\mathbf{A}}) - k - \ln\left(\frac{\det\Sigma_{\mathbf{A}}}{\det\Sigma_{\mathbf{B}}}\right)\right) \tag{6.3.3}$$

Where μ is the mean of the points in the corresponding lick, and Σ is the covariance. This value $D_{\mathrm{KL}}(\mathcal{N}_{\mathbf{A}}\|\mathcal{N}_{\mathbf{B}}) \geqslant 0$ can be thought of as the distance of lick $\mathbf{B}$ from lick $\mathbf{A}$, but the Kullback-Leibler divergence is not a true metric because it is not symmetric. This is not a problem, because it is only applied in one direction – in fact, the motion of flames in general does not look the same forwards and backwards, so a non-symmetrical measure is entirely acceptable. Whether the asymmetry is actually accurate to the movement however, is not important, because the licks are sufficiently distinct in the data to make the matching clear. A simpler measure (such as the Bhattacharyya distance [Bha46]) would likely work as well.

So the estimate for the motion between each pair of elements from the original flames $\mathcal{F}_A$ and $\mathcal{F}_B$ is high if the divergence is low and vice versa. The exact weights are scaled as follows assuming that the sum over all possibilities of $\mathbf{A}$ equals one (i.e. every lick evolves into another or dies out). For $i = 1, \ldots, n$ representing licks in $\mathcal{F}_A$, and $j = 1, \ldots, m$ representing licks in $\mathcal{F}_B$, the motion estimate between licks i and j is given by

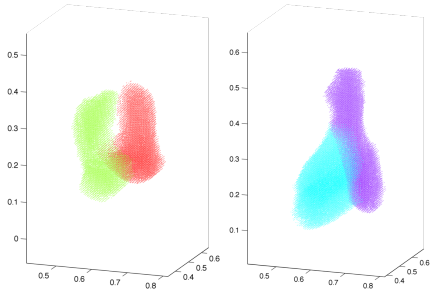
$$M_{i\emptyset} = k \tag{6.3.4}$$

$$M_{ij} = \frac{\exp(-d_{ij})}{k + \sum_x \exp(-d_{ix})} \tag{6.3.5}$$

where d_{ij} is the Kullback-Leibler divergence from lick $\mathbf{A}_i$ to $\mathbf{B}_j$. k is a small constant that represents a cut-off, in some cases it will be more likely for a lick to die out than significantly move or change size. The value of k was found heuristically for the data, but was not very sensitive – usually a dying lick of flame is obviously not related to the other licks, the method just needs another alternative. This gave suitable results so was considered sufficient, rather than applying something like Laplace smoothing [MRS08].

The matching is obtained by looking for the maximum probability match from flame $\mathcal{F}_A$ for each lick in flame $\mathcal{F}_B$. Formally,

$$\left(\underset{x}{\mathrm{argmax}}(M_{xj}),\ j\right) \text{ for all } j. \tag{6.3.6}$$



$M =$

	cyan	purple	$\emptyset$
red	0.47	0.48	0.05
green	0.73	0.19	0.08

	cyan	purple
argmax {red,green}	green	red

Figure 6-3: An example of the matrix of weights produced when calculating matching

The reason for choosing the most likely licks of $\mathcal{F}_A$ for each lick in $\mathcal{F}_B$, rather than vice-versa, is because it was found that modelling licks disappearing was much more useful than appearing. While in some cases the reverse may be useful, in particular if there is an 'off-screen' source of flame that is generating more, this did not occur in the data studied. This also means that licks can split, however, they will not match as merging (because each lick in $\mathcal{F}_B$ can only have one corresponding lick in $\mathcal{F}_A$). In practice it was decided that having one lick evolve and one lick die out would be a close enough proxy for merging. However there are alternative strategies for dealing with merging flames, such as considering the weights from the power set of licks of $\mathcal{F}_A$ rather than just individually, or simply looking for cases where two or more licks had similar weights.

The matching process is illustrated in Figure 6-3, and example results are given in Figure 6-4.

6.4 Control

One of the key objectives of this work is intuitive editability. The most significant advantage of the methods presented is that they can be controlled easily, and this is true of the generative model in this chapter as well. The above system is straightforward, it is possible to control the random variation, and this will have an impact on how erratic the motion of the flame is (with the risk of becoming unflamelike). Other than this however, one training set produces one new flame, albeit with some random variation between results. Two other methods for controlling the result were developed, which are detailed

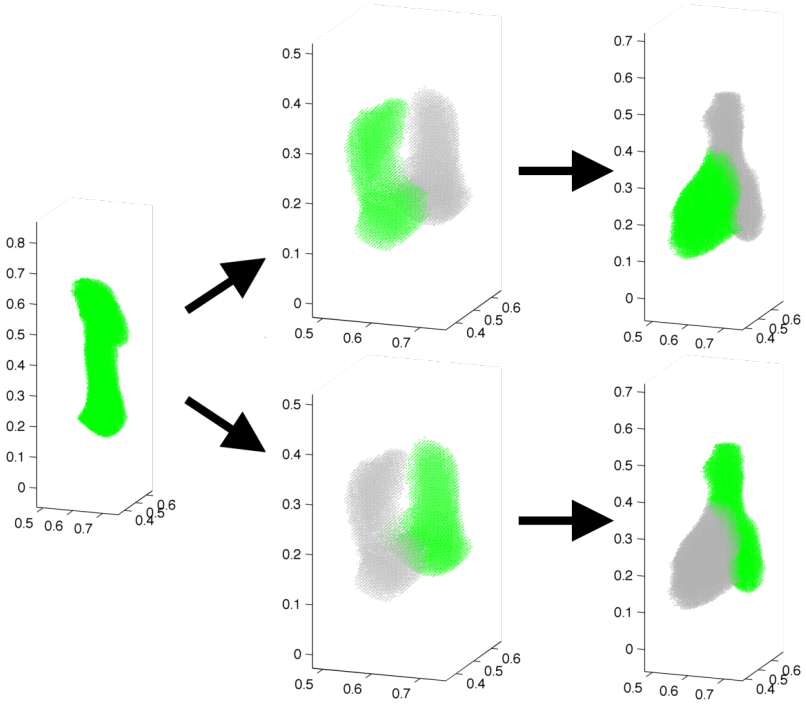


Figure 6-4: Two consecutive examples of complex flame matching

here, and some other ideas are included as well.

6.4.1 Direct Modification

A simple way to control the results of the generative method is to modify the underlying training set according to some desired visual change. Changes to a single flame can be learned and then applied to the entire parameter space. For example, a transformation can be applied to the core of every flame to make the flame taller:

$$c^*(u) = M \begin{bmatrix} c(u) \\ 1 \end{bmatrix} \tag{6.4.1}$$

using homogeneous coordinates to represent the points along the core. In this case, the matrix M could be the following:

$$M = T_{-c(1)} S_{(1,1,1.5)} T_{c(1)} \tag{6.4.2}$$

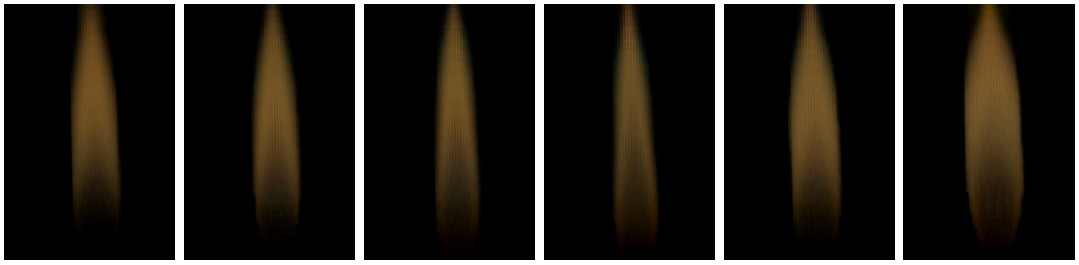


Figure 6-5: A flame controlled to be particularly tall

where

$$T_{\mathbf{v}} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S_{\mathbf{v}} = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4.3}$$

i.e., a translation based on the base point of the flame, a scaling in the positive z-direction, and then a translation back.

A tall flame, generated from a modified version of the regular candle dataset, is shown in Figure 6-5. Other simple transformations to any parameter, including the density parameters, can be made in a similar manner: making the entire video brighter, changing the width of the flame, increasing the contrast due to the glowing soot particles, and so on. Even non-affine transformations may be applied, although obviously care must be taken to preserve continuity in the motion of the flame and the flame-like appearance.

It is also possible to modify the matching data from Section 6.3 to speed up or slow down the animation. The flames shown here, the candle and the lighter, were both filmed at one hundred frames per second. Since this is faster than most conventional animations, it was found to be effective to modify the matching data in the following way:

$$((1,2),(2,3),(3,4),\ldots,(n,m)) \rightarrow ((1,5),(2,6),(3,7),\ldots,(n,m+3)) \tag{6.4.4}$$

this simulated a path that was the same, but that moved much faster. This change specifically reduces the frame rate from one hundred to twenty five frames per second. This gives the advantage of a faster processing time, an animation which is still suitable for viewing purposes, and the underlying data of much higher frame rate. By interpolating points between frames in the various parameter spaces, it would be possible to generate videos of any frame rate, even one that is slower than the original. The interpolation could

be weighted on the density estimation (as in the actual generation), which could lead to better results than taking a simple linear motion between the two frames.

6.4.2 Control by Example

One frequently used method of control in this work has been to use example data, for example, data captured from real world video. In some situations, tuning parameters is much more difficult than just providing an example, especially when these already exist. This technique can be used on the generative process as well; meaning a training set from one flame can be used to animate another flame of differing motion or appearance.

This is particularly useful to generate something that did not previously exist, such as a single flame with no motion. In other words, assume there exists a sequence of flames as previously (the training set), and a single target flame (the example). Here, a flame that was fitted to a single 2D image of a pink lighter (from Section 5.4) will be used, along with the training data from the regular yellow lighter. The aim is to copy the pink appearance of the example flame onto the motion and shape of the training set.

The colour parameters for each flame in the training set, s and u, are modified in the same fashion as the previous section on direct control. As in the generation step, assume each parameter is discretised along the length of the core. Thus, for flame i in the training set, assume its parameters are denoted $\mathbf{s}_i$ and $\mathbf{u}_i$, and the parameters of the example flame are $\mathbf{s}_\mathcal{E}$ and $\mathbf{u}_\mathcal{E}$. Then new parameters for the modified training set are

$$\mathbf{s}'_i = \mathbf{s}_i + \mathbf{v} \tag{6.4.5}$$

where $\mathbf{v}$ is given by

$$\mathbf{v} = (\mathbf{s}_\mathcal{E} - \mathbf{s}_k),\ k = \underset{i=1,2,\ldots}{\operatorname{argmin}} \|\mathbf{s}_i\| \tag{6.4.6}$$

i.e. the vector between the example flame and the flame in the training set that is closest to the origin. This is to prevent the values in the training data from becoming negative as much as possible after the transformation. It is still possible that some will, and in this case they are simply set to zero.

The results of this example are shown in Figure 6-6.

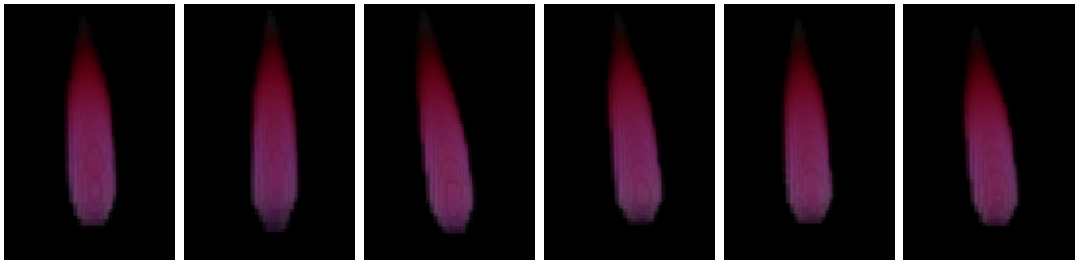


Figure 6-6: Combining the pink lighter flame with the motion from the regular lighter

6.4.3 Future Work

While not developed to the point that gives good enough results, some ideas for future methods of control have been considered.

It might be possible to control the motion not by modifying the underlying training data, but the generation process itself. Firstly assume that there exists a large training set that encompasses a range of motion. This can be from one long video, or the information from several videos concatenated. The candle dataset is one such example: it was filmed without any interference for half of the video's length, and for the second half varying levels of breeze were applied from one side.

Then it is possible to control the motion estimate step of the generative process to weight towards a particular appearance of a flame. For example, weighting towards an uninterrupted flame, even with erratic motion from a breeze being present in the data, would result in a flame that did not exhibit much flickering. Or weighting towards a flickering flame could simulate a flame in a constant wind. A naive implementation of this idea created a problem between the regular motion estimate and the weighting. It could be beneficial to stop weighting towards the goal point once the animation has reached it, allowing the regular motion to move away, before reapplying the weighting. By moving the weight point, different 'cycles' in the flame animation can be picked out. By turning on and off this feature at different points along the animation, it should be possible to create the realistic motion of moving between states. This is extremely useful to an animation pipeline, as it means that the flame can be made to synchronise with other events in a scene, such as a gust of wind.

Another idea would be to create interaction between the flames being generated and an object. This (possibly moving) object could deform the core of the licks, wrapping the fire around the object. This is being actively considered in the next stages of this work.

6.5 Conclusion

This chapter presented a method for taking a sequence of flames and using these to create a completely new sequence which looks and moves similarly to the original, but does not share any exact frames. This, along with the material in previous chapters, means that it is possible to take video data of a flame and generate new flames of the same style. Furthermore, the control available in this process means that it can also create distinctly different looking videos, by making a few parameter changes or even by just providing another flame example. With access to a library of flame videos, such as the one in development detailed in Chapter 3, it is possible to take a single two-dimensional image, and use this chapter's contributions to make an animated three-dimensional flame that retains the appearance of the image. This is something which is unthinkable with previous techniques.

The system can also be used to generate animations that are identical to the original but at a different frame rate. Currently this has been effectively used to speed up video. It could also be used to slow it down, which would require interpolating points along the motion path. The system could take nearby points into consideration to make sure the flame is realistic, which should provide a more convincing result than simply blending between frames of the original video.

Note that the generation algorithm does not generate flames that split, merge, or extinguish. An approach was presented for capturing this sort of motion from complex flames, however due to technical limitations with the data, not enough usable data could be extracted, hence this is not incorporated into the algorithm. Given better input data, it would be possible to use this to drive a statistical model of flames exhibiting this complex motion.

Chapter 7
Conclusion

This chapter presents a conclusion of the work in the rest of the document. First the claims and objectives from the introduction are evaluated against the work that has been completed, and novel contributions of the work are highlighted. Next the limitations of the work are considered, and strategies for addressing any shortcomings will be discussed. Finally, some future work in the area will be proposed.

7.1 Contributions

The hypothesis asserted at the start of this document was: *it is possible to generate three-dimensional, intuitively editable, dynamic models of flames from video.* The intervening chapters have substantiated this claim, met the objectives, and many notable contributions have been made in the process.

- Chapter 3 detailed the process of acquiring three-dimensional models from video. Both pre-existing data, and also a new dataset, which is state of the art of its type.
- Chapter 4 presented the novel 'flame core' model for flames. This model produces realistic looking flames, that are also intuitively editable. Examples of control are given where parameters correspond intuitively to results.
- Chapter 5 showed that it is possible to fit a flame core model to the standard non-editable three-dimensional models acquired in Chapter 3, thereby completing the objective to have intuitively editable flames from video. It also showed how some parameters could be fit to a single uncalibrated view, which is novel in flame reconstruction literature.

- Chapter 6 used flame core models acquired from real data to create brand new, unique flame animations. This shows that from a single video, it is possible to generate novel animations using the visual and motion information captured with the flame core model. This trivialises the process of producing new flame content. Furthermore, the process offers intuitive controls of its own, meaning new animations can be visually distinct.

7.2 Evaluation and Future Work

The main contribution of the work is the flame core model, and its ability to be driven by video data. Compared to previous models (detailed in Chapter 2) it offers several improvements. The foremost being its intuitive editability, which has been discussed in Chapter 4, and the ability to generate new animations based on an example sequence, from Chapter 6. These are both distinct advantages over previous methods, both those that do not use video as input, and those that do.

Obviously being drivable by video input is another advantage compared to methods that are not. However the results of the flame core model are not always as visually impressive. While small simple flames like candles and lighters are very realistic, work such as structural modelling (Section 2.2.5) can generate larger dynamic flames such as bonfires or even a fire-breathing dragon. They also have more dynamic effects such as flickering and turbulence. This work in particular is interesting because it shares a similar concept. It is proposed that it would be possible to convert flames between the two models, giving the best of both worlds: manual and video control of the flame core method, and the graphical appearance of structural modelling. Structural modelling rotates a profile around a curve, and then fills this volume with particles. These particles could instead be generated based on a flame core's parameters, and then be subject to the flickering and turbulence of the original method. Similarly, flame core models could be fit to particles generated by structural modelling, to convert in the other direction.

As shown in Section 5.5.1, when compared to previous reconstruction work, the flame core model has mixed results. It is editable, which the other methods are not. In terms of fitting to input data, it performs quite well when fitting to the candle and lighter datasets, but not as well on the alcohol dataset. To be specific, image-based tomography [IM04] outperforms it visually and quantitatively, which is expected since this work is used to produce the models to which the flame core model is fitted. For the alcohol dataset, flame-

sheet decomposition [HK03] generates better looking results from the original camera viewpoints, and when interpolating between them. This is likely true for the other datasets as well, however the flame core model produces a full three-dimensional density, and so would perform better when viewed from above, and may be a closer match to the true density (but this is only speculation as the method was not reproduced). In the single-lick datasets, the greatest error occurs when fitting to flames that are radially non-symmetric, and it is possible this could be improved using a non-symmetric density function, possibly a von Mises-Fisher mixture model [DS03a, BDG$^+$05]. Finally the flame core model has the extra benefit of being more space-efficient compared to voxel-based methods, because of the reduction in dimensionality (approximately 98% smaller).

The flame core is the first known work to show results of fitting to a single uncalibrated view of a flame, as in Chapter 5. Previous reconstruction literature has simply stated as fact that multiple views are required to reconstruct a flame. This is probably true if the goal is a perfectly faithful representation. However, when a human views a video of a flame, they can easily interpret a three-dimensional phenomenon. It may be that some optical illusion or lack of information is causing a slightly different interpretation from the ground truth, but it still looks like a real flame. The same philosophy can be applied to reconstruction: provided the three-dimensional result looks realistic from all angles, and looks like the input video from the matching angle, then it is a useful result. The flame core model provides a good foundation for this work, as the density parameters can already be interpreted from a single view. Provided a collection of similar looking flames exists, their parameters can be modified to generate a flame that looks similar to the single view input, as shown in Chapter 6. This process could be further tuned to the point where the appearance and motion can match a single view exactly. Alternatively, existing flames can be used as priors to intelligently reconstruct the depth of flames from the single video, specifically the depth of the core.

Another idea for future work would be to incorporate interaction with external events into the flame core model. In Chapter 6 a primitive system for responding to changes in the environment on cue was proposed, this can already be achieved with the current methods. It would be interesting to incorporate the effect of an object physically interacting with the volume of the flame. This could be achieved by warping the density field around the object at the time of rendering, possibly by modifying the core of the flame to have some thickness defined by the object, and having the flames form around this. The model could also be used to drive other effects in a scene: a simple example would be to emit smoke or embers. In the case of smoke, this could be paired with smoke found from video [IM06],

generated via particles on the surface of the flame, or use another controllable simulation method [FSJ01, YCZ11, FL04].

Finally, this project saw the acquisition of two new datasets of fire videos, a candle and a lighter, which are detailed in Chapter 3. These are filmed with enough cameras at a high enough quality to create good quality three-dimensional reconstructions, which is significant for this type of research as there is only one other source of this data, which is technologically outdated in comparison. Unfortunately due to safety restrictions only these simple flames could be filmed in the course of the project. The previous data was of a plate of alcohol; this type of flame creates far more complicated flame shapes and would give an opportunity to model more complicated motion including splitting, merging, and extinguishing. However the recorded data is not of high enough quality to extract this information. The flame core model was designed with some of these elements in mind: it supports more than one core per flame, the fitting algorithm can attempt to split complicated flames, and there is a method for analysing motion of these flames. It would be of great interest to acquire new videos of more complicated flames, and try to test these techniques in their entirety. It is likely that even more developments to the generation process (Chapter 6) could be made, for example to offer control over the apparent turbulence of the flame. This would widely expand the types of flame animations that could be made, and generally improve the realism of the flame outputs considerably.

Appendices

Appendix A
Visually Intuitive Parameters for the Gamma Distribution

In Chapter 4 different shapes for the density function were considered, based on the distribution functions of the normal, gamma, and Weibull distributions. The normal distribution function, also called a Gaussian function, was chosen to use in this work. One reason is that the normal distribution's parameters more intuitively control the shape of the curve. However, while far less convenient, it is possible to reparameterise the gamma distribution function so that its parameters control the curve in the same way. This appendix details this process, though ultimately the normal distribution was chosen due to better fit to real data.

The process described can also be applied to the Weibull density function, however solving the resulting equation requires using a numerical estimate method, and so is inexact and more computationally complex.

A.1 Intuitive Parameters

In Section 4.2, the most intuitive parameters for editing were identified as the brightness at the peak, the position of the edge of the flame, and the width of the fall-off from the peak – the fuzziness of the flame. These correspond to the maximum value, the mode, and the 'variance' of the distribution respectively. Note that since the shape of the function is all that is required, the normalising constant of the probability density function will be dropped. This means that the distribution loses its probabilistic meaning, and is instead just used for visual shape.

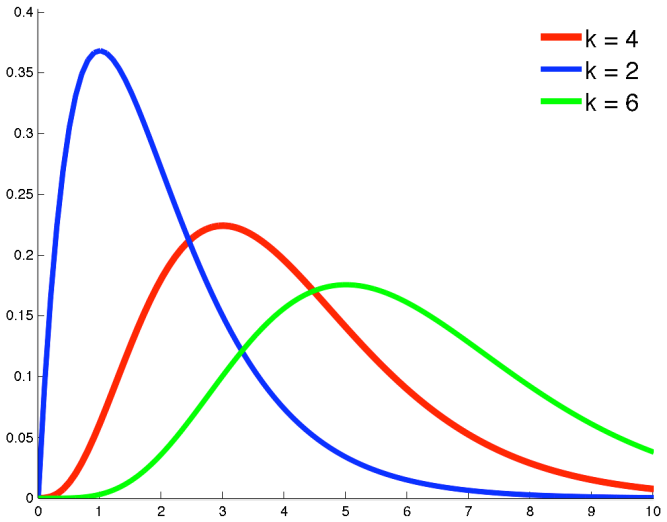


Figure A-1: The effect on the density distribution when the shape parameter varies

A.2 Gamma Distribution

The probability density function for the gamma distribution, without its normalisation constant, is:

$$f(x|k,\theta) = x^{k-1}e^{-\frac{x}{\theta}} \tag{A.2.1}$$

This results in curves of wildly differing shapes, as with the normalised version of the function, shown in Figure A-1. The goal is to make a form of control with parameters that govern the width (variance) of the function, the position of the maximum point, and and overall scale (height). Call these w for width, m for maximum (or mode), and s for scale.

Since the only effect of removing the normalisation constant is a (vertical) scale factor, the closed form solutions for the mode and variance for the gamma distribution still give the visually expected results. The function reaches its maximum at the mode (by definition), which is given by:

$$m = (k-1)\theta \tag{A.2.2}$$

Hence, it is possible to scale the function in Equation A.2.1 by $f(m)$:

$$f'(x|k,\theta,s) = s\frac{x^{k-1}e^{-\frac{x}{\theta}}}{(k-1)^{k-1}\theta^{k-1}e^{1-k}} \tag{A.2.3}$$

The scale factor s now controls the height of the function, as desired.

Next, controlling the spread (variance) and the distance of the brightest point (the mode). The variance is given by

$$w = k\theta^2, \tag{A.2.4}$$

hence

$$\theta^2 = \frac{k}{w}. \tag{A.2.5}$$

Using the equation for the mode (Equation A.2.2),

$$\theta = \frac{m}{(k-1)}. \tag{A.2.6}$$

Note it is assumed that $m > 0$, and by definition $\theta > 0$ and so $k > 1$.

Then from equations A.2.5 and A.2.6,

$$\frac{w}{k} = \frac{m^2}{(k-1)^2}, \tag{A.2.7}$$

$$\frac{(k-1)^2}{k} = \frac{m^2}{w}, \tag{A.2.8}$$

$$\frac{k^2-2k+1}{k} = \frac{m^2}{w}, \tag{A.2.9}$$

$$\frac{k^2+1}{k} = \frac{m^2}{w} + 2, \tag{A.2.10}$$

$$k^2 - \left(\frac{m^2}{w} + 2\right)k + 1 = 0, \tag{A.2.11}$$

which is a quadratic in terms of k. Solving this:

$$k = \frac{\left(\frac{m^2}{w}+2\right) \pm \sqrt{\left(-\frac{m^2}{w}-2\right)^2 - 4}}{2}, \tag{A.2.12}$$

$$k = 1 + \frac{m^2}{2w} \pm \frac{\sqrt{\left(\frac{m^2}{w}\right)^2 + 4\frac{m^2}{w}}}{2}. \tag{A.2.13}$$

For clarity of notation introduce

$$t = \frac{m^2}{w}, \tag{A.2.14}$$

then

$$k = 1 + \frac{1}{2}\left(t \pm \sqrt{t^2 + 4t}\right). \tag{A.2.15}$$

It is desirable to just have one solution. Since it is assumed that $k > 1$ (from Equation A.2.6)

$$t \pm \sqrt{t^2 + 4t} > 0, \tag{A.2.16}$$

thus for the negative case

$$t > \sqrt{t^2 + 4t}, \tag{A.2.17}$$

and since $t > 0$ from Equation A.2.14

$$t^2 > t^2 + 4t, \tag{A.2.18}$$

which is a contradiction. So only the positive case is valid, and the formula for k is

$$k = 1 + \frac{1}{2}\left(t + \sqrt{t^2 + 4t}\right). \tag{A.2.19}$$

Thus, for any parameters w and m, the k and θ parameters for Equation A.2.3 can be found, first using Equation A.2.19 for k, and then Equation A.2.6 for θ. These equations could replace the k and θ parameters so that Equation A.2.3 could be written entirely in terms of w and m, however the resulting function would clearly not be readable in print.

Figure A-2 shows the effect of changing these new parameters.

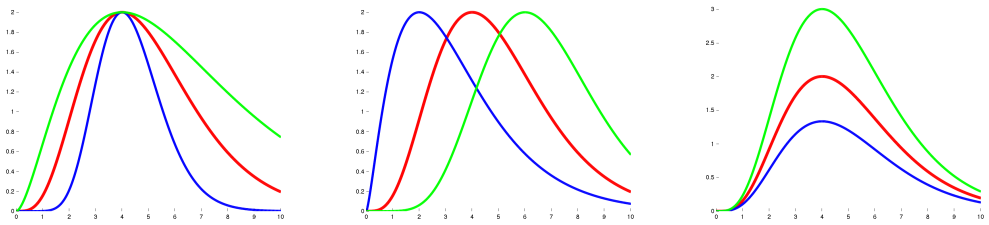


Figure A-2: The effect of changing the new parameters on the density
From left to right: w, m, s

Bibliography

[AW10] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[BDG+05] Arindam Banerjee, Inderjit S Dhillon, Joydeep Ghosh, Suvrit Sra, and Greg Ridgeway. Clustering on the unit hypersphere using von mises-fisher distributions. *Journal of Machine Learning Research*, 6(9), 2005.

[Bha46] Anil Bhattacharyya. On a measure of divergence between two multinomial populations. *Sankhyā: The Indian Journal of Statistics*, pages 401–406, 1946.

[Bjö96] Ake Björck. *Numerical methods for least squares problems*. Siam, 1996.

[Blu67] H. Blum. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967.

[BRGIG+14] M Balsa Rodríguez, E Gobbetti, JA Iglesias Guitián, M Makhinya, F Marton, R Pajarola, and SK Suter. State-of-the-art in compressed gpu-based direct volume rendering. In *Computer Graphics Forum*. Wiley Online Library, 2014.

[BS09] A. Basharat and M. Shah. Time series prediction by chaotic modeling of nonlinear dynamical systems. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1941–1948. IEEE, 2009.

[BST05] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven automatic centerline extraction. *Medical Image Analysis*, 9(3):209–221, 2005.

[Cao97] Liangyue Cao. Practical method for determining the minimum embedding dimension of a scalar time series. *Physica D: Nonlinear Phenomena*, 110(1):43–50, 1997.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[CM10] John A Conkling and Chris Mocella. *Chemistry of pyrotechnics: basic principles and theory*. CRC press, 2010.

[CR74] Edwin Catmull and Raphael Rom. A class of local interpolating splines. *Computer aided geometric design*, 74:317–326, 1974.

[DB78] Carl De Boor. *A practical guide to splines*, volume 27 of *Applied Mathematical Sciences*. Springer, New York, NY, 1978.

[DCH88] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *ACM Siggraph Computer Graphics*, volume 22, pages 65–74. ACM, 1988.

[DCWS03] Gianfranco Doretto, Alessandro Chiuso, Ying Nian Wu, and Stefano Soatto. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, 2003.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[DS03a] Inderjit S Dhillon and Suvrit Sra. Modeling data using directional distributions. *Department of Computer Sciences. University of Texas Technical Report TR-03-06*, 2003.

[DS03b] Gianfranco Doretto and Stefano Soatto. Editable dynamic textures. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–137. IEEE, 2003.

[FL04] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 441–448. ACM, 2004.

[FS86] Andrew M Fraser and Harry L Swinney. Independent coordinates for strange attractors from mutual information. *Physical review A*, 33(2):1134, 1986.

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 2001.

[GS97] N. Gagvani and D. Silver. Parameter controlled skeletonization of three-dimensional objects. *TechnicalReport CAIP*, 1997.

[Han98] P.C. Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion.* Number 4. Society for Industrial Mathematics, 1998.

[Has02] S.W. Hasinoff. Three-dimensional reconstruction of fire from images, 2002.

[HK03] Samuel W. Hasinoff and Kiriakos N. Kutulakos. Photo-consistent 3d fire by flame-sheet decomposition. *Computer Vision, IEEE International Conference on*, 2:1184, 2003.

[HK07] S.W. Hasinoff and K.N. Kutulakos. Photo-consistent reconstruction of semi-transparent scenes by density-sheet decomposition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(5):870–885, 2007.

[HMS+00] T.C. Henderson, P.A. McMurtry, P.J. Smith, G.A. Voth, C.A. Wight, and D.W. Pershing. Simulating accidental fires and explosions. *Computing in Science & Engineering*, 2(2):64–76, 2000.

[Hot33] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[HS89a] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, pages 502–516, 1989.

[HS89b] William Hibbard and David Santek. Interactivity is the key. In *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, pages 39–43. ACM, 1989.

[HS91] Robert M Haralock and Linda G Shapiro. *Computer and robot vision.* Addison-Wesley Longman Publishing Co., Inc., 1991.

[IKL+10] I. Ihrke, K.N. Kutulakos, H. Lensch, M. Magnor, and W. Heidrich. Transparent and specular object reconstruction. In *Computer Graphics Forum*, volume 29, pages 2400–2426. Wiley Online Library, 2010.

[IM04] I. Ihrke and M. Magnor. Image-based tomographic reconstruction of flames. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 365–373. Eurographics Association, 2004.

[IM06] I. Ihrke and M. Magnor. Adaptive grid optical tomography. *Graphical Models*, 68(5):484–495, 2006.

[Jol02] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[KLS11] Matej Kristan, Aleš Leonardis, and Danijel Skočaj. Multivariate online kernel density estimation with gaussian kernels. *Pattern Recognition*, 44(10):2630–2642, 2011.

[Knu69] Donald Ervin Knuth. *The Art of Computer Programming: Seminumerical Algorithms. II.* Addison-Wesley, 1969.

[Lau94] A. Laurentini. The visual hull concept for silhouette-based image understanding. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(2):150–162, 1994.

[Lev44] Kenneth Levemberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[Lev88] Marc Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8(3):29–37, 1988.

[LF02] A. Lamorlette and N. Foster. Structural modeling of flames for a production environment. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 729–735. ACM, 2002.

[LKC94] Ta-Chih Lee, Rangasami L Kashyap, and Chong-Nam Chu. Building skeleton models via 3-d medial surface axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56(6):462–478, 1994.

[Mah36] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[Mar63] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

[MBGN98] Tom McReynolds, David Blythe, Brad Grantham, and Scott Nelson. Advanced graphics programming techniques using opengl. *Siggraph 1998 Course Notes*, 1998.

[MH99] Tomas Möller and John F Hughes. Efficiently building a matrix to rotate one vector to another. *Journal of graphics tools*, 4(4):1–4, 1999.

[MR98] Ramakrishnan Mukundan and KR Ramakrishnan. *Moment functions in image analysis: theory and applications*, volume 100. World Scientific, 1998.

[MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[NFJ02] D.Q. Nguyen, R. Fedkiw, and H.W. Jensen. Physically based modeling and animation of fire. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 721–728. ACM, 2002.

[Ots75] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[Pal99] Stephen E Palmer. *Vision science: Photons to phenomenology*, volume 1. MIT press Cambridge, MA, 1999.

[Per01] Karl Person. On lines and planes of closest fit to systems of points in space. *Philosophical magazine*, 2(6):559–572, 1901.

[PN+01] Ken Perlin, Fabrice Neyret, et al. Flow noise. In *28th International Conference on Computer Graphics and Interactive Techniques (Technical Sketches and Applications)*, 2001.

[PP06] V. Pegoraro and S.G. Parker. Physically-based realistic fire rendering. In *Proc. Eurographics Workshop on Natural Phenomena*, pages 51–59, 2006.

[Rad17] J. Radon. Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten. *Akad. Wiss.*, 69:262–277, 1917.

[Ree83] W.T. Reeves. Particle systems: A technique for modeling a class of fuzzy objects. *ACM Transactions on graphics*, 2(2):91–108, 1983.

[RWPW98] Paul D Ronney, Ming-Shin Wu, Howard G Pearlman, and Karen J Weiland. Experimental study of flame balls in space: Preliminary results from sts-83. *AIAA journal*, 36(8):1361–1368, 1998.

[Sch96] A Schwarz. Multi-tomographic flame analysis with a schlieren apparatus. *Measurement Science and Technology*, 7(3):406, 1996.

[SF93] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 369–376. ACM, 1993.

[SF95] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 129–136. ACM, 1995.

[SGF+12] Gregory P. Smith, David M. Golden, Michael Frenklach, Nigel W. Moriarty, Boris Eiteneer, Mikhail Goldenberg, C. Thomas Bowman, Ronald K. Hanson, Soonho Song, William C. Gardiner Jr., Vitali V. Lissianski, and Zhiwei Qin. Gri-mech 3.0, 2012.

[Sil86] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[SM05] Timo Stich and Marcus Magnor. Learning flames. In *Vision, Modeling, and Visualization 2005: Proceedings, November 16-18, 2005, Erlangen, Germany*, page 65. IOS Press, 2005.

[SMP05] Tomáš Svoboda, Daniel Martinec, and Tomáš Pajdla. A convenient multicamera self-calibration for virtual environments. *PRESENCE: teleoperators and virtual environments*, 14(4):407–422, 2005.

[SSSE00] A. Schödl, R. Szeliski, D.H. Salesin, and I. Essa. Video textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 489–498. ACM Press/Addison-Wesley Publishing Co., 2000.

[TZCO09] A. Tagliasacchi, H. Zhang, and D. Cohen-Or. Curve skeleton extraction from incomplete point cloud. In *ACM Transactions on Graphics (TOG)*, volume 28, page 71. ACM, 2009.

[Urb06] Peter Urben. *Bretherick's Handbook of Reactive Chemical Hazards: 2-Volume Set*. Academic Press, 2006.

[Woo80] R.J. Woodham. Photometric method for determining surface orientation. *Optical engineering*, 1(7):139–144, 1980.

[WT90] Geoff Wyvill and Andrew Trotman. *Ray-tracing soft objects*. Springer, 1990.

[YCZ11] Zhi Yuan, Fan Chen, and Ye Zhao. Pattern-guided smoke animation with lagrangian coherent structure. In *ACM Transactions on Graphics (TOG)*, volume 30, page 136. ACM, 2011.

[ZRL77] GW Zack, WE Rogers, and SA Latt. Automatic measurement of sister chromatid exchange frequency. *Journal of Histochemistry & Cytochemistry*, 25(7):741–753, 1977.